PROGRAMMEZ.COM

PROGRAMMEZ!

Mensuel n°191 - Décembre 2015

le magazine du développeur

Choisir son école d'informatique

Open Data

Où trouver les données ouvertes ?

© maxsattana

Développer et intégrer facilement les open data

Les nouvelles architectures logicielles

1ere partie : les microservices, Java

Sport Geek

Montez et codez votre rameur connecté !

Développer pour Android, iOS, Windows

Xamarin

Focus sur : Xamarin.Mac, Xamarin.Forms, Active Directory, Trucs & astuces

Kinect

Développer avec le SDK Kinect 2

Navigateur

Découvrir les Web Worker

M 04319 - 191 - F: 5,95 € - RD

Printed in EU - Imprimé en UE - BELGIQUE 6,45 € SUISSE 12 FS - LUXEMBOURG 6,45 € - DOM Surf 6,90 € Canada 8,95 $ CAN - TOM 940 XPF - MAROC 50 DH

COMMANDEZ WINDEV 21

OU WEBDEV 21 OU WINDEV MOBILE 21

ET RECEVEZ

LE NOUVEL iPhone 6 S

Prolongation exceptionnelle jusqu'au **18 Décembre**

iPhone 6 S Plus

iPhone 6S Plus **64GB**.

Choix de la couleur sur le site

ou

iPhone 6 S

iPhone 6S **128GB**.

Choix de la couleur sur le site

Ou choisissez parmi:

- 1x **iPad Air 2** Wi-Fi+Cellular 128GB
- 2x **iPhone 5S** 16GB
- 2x **iPad Mini 4** Wi-Fi 64GB
- 2x **iPad Air 2** Wi-Fi 16GB...

D'autres matériels sont proposés sur le site www.pcsoft.fr

Aucun abonnement à souscrire. Compatible tous opérateurs

OPÉRATION POUR 1 EURO DE PLUS

Pour bénéficier de cette offre exceptionnelle, il suffit de commander WINDEV 21 (ou WINDEV Mobile 21, ou WEBDEV 21) chez PC SOFT au tarif catalogue avant le 18 Décembre 2015. Pour 1 Euro de plus, vous recevrez alors le ou les magnifiques matériels que vous aurez choisis. Offre réservée aux sociétés, administrations, mairies, GIE et professions libérales, en France métropolitaine. L'offre s'applique sur le tarif catalogue uniquement. Voir tous les détails et des vidéos sur : www.pcsoft.fr ou appelez-nous (04.67.032.032).
Le Logiciel et le matériel peuvent être acquis séparément. Tarif du logiciel au prix catalogue de 1.650 Euros HT (1.973,40 TTC). Merci de vous connecter au site www.pcsoft.fr pour consulter la liste des prix des matériels et les dates de disponibilité. Tarifs modifiables sans préavis.

www.pcsoft.fr

Elu «Langage le plus productif du marché»

Fournisseur Officiel de la Préparation Olympique

PROGRAMMEZ!
le magazine du développeur

sommaire

Un code, de multiples possibilités.

Nous vous parlons régulièrement des outils et plateformes de développement permettant de coder une fois, et de cibler plusieurs environnements. Dans le mobile, c'est un enjeu important. Mais, il ne faut pas se dire qu'un tel outil fera tout et tout seul. Absolument pas. Si le code back-end sera largement compatible, l'interface et un x % de ce code seront à reprendre, à adapter. L'interface n'est pas à négliger. Et d'autre part, il faut aussi une certaine réactivité de la plateforme de développement pour corriger les bugs, supporter les nouvelles API et suivre les évolutions des systèmes mobiles.

Xamarin est sans doute l'environnement le plus connu dans ce domaine. Même s'il n'est pas parfait, et que les défauts de certains outils sont souvent évoqués par les communautés, reconnaissons le travail accompli par l'éditeur. Ce mois-ci, nous revenons sur quelques fondamentaux de l'outil.

Nous parlerons aussi d'un autre sujet chaud : les nouvelles architectures logicielles, tout le moins, quelques-unes d'entre elles. Le sujet est important, car elles vont directement impacter vos développements et la manière de penser, de structurer et de déployer vos applications. Nous aurons l'occasion d'y revenir dans les prochains mois. Ce sujet ne fait que débuter.

Nous parlerons aussi, concrètement, d'Open Data avec un peu de théorie (il faut bien) et surtout des exemples très réels pour bien comprendre l'usage des données ouvertes.

Autre sujet qui va intéresser nos jeunes lecteurs et les parents, le choix d'une école d'informatique. Le thème peut paraître classique et éculé, mais à la rédaction nous nous sommes posé la question sur les critères de choix, et, finalement, comment s'orienter dans la multitude de filières, de formations et de diplômes. Ce choix ne doit pas être fait à la légère, car il va définir votre futur...

Avec quelques semaines d'avance, nous vous souhaitons de bonnes fêtes de fin d'année (eh oui déjà !).

Le boss
ftonic@programmez.com

À lire dans le prochain numéro n°192 en kiosque le 31 décembre 2015

Retour vers le passé

Atari ST, Amiga, Apple 2, Amstrad CPC.. cela vous dit quelque chose ? Non ? Faisons un grand bond de 30 ans en arrière ! Un voyage surprenant dans la programmation « oldschool » et la Démoscène française !

Cahier spécial Drupal 8

La version finale de Drupal 8 est enfin arrivée. Un guide complet pour bien démarrer !

1/10

SurfaceBook et Macbook même combat dans la réparation selon iFixit

Apple

ouvre des librairies de cryptos, https://developer.apple.com/cryptography/

Les extensions arriveront en 2016 dans

Edge

Cortana sur iOS,

une bêta privée disponible

Google

va-t-il produire son propre processeur ?

Windows 3.1 dépassé ?

L'aéroport d'Orly l'utilise toujours.

Qui investit le plus en Recherche & Développement* ?

+ 13 milliards Samsung	+ 10 milliards Intel	+ 10 milliards Microsoft	+ 8 milliards Google

Longtemps très loin derrière, **Apple** a renforcé son budget de R&D : 8 milliards estimés cette année...

** données Fortune.com*

Firefox sans plug-ins

en 2016

L'art du pliage des

Nexus 6P

Tesla avait intégré par mise à jour, et quelques capteurs, la conduite autonome...

pour la limiter quelques jours plus tard...

Index Tiobe : Objective-C out, Java au top

Nov 2015	Nov 2014	Tendance	Langage	%	Evolution
1	2	^	Java	20.403%	+6.01%
2	1	v	C	17.145%	-0.32%
3	4	^	C++	6.198%	+0.10%
4	5	^	C#	4.318%	-0.67%
5	7	^	Python	3.771%	+1.18%
6	6		PHP	3.248%	+0.20%
7	8	^	JavaScript	2.473%	+0.38%
8	10	^	Visual Basic .NET	2.223%	+0.16%
9	14	^^	Ruby	2.038%	+0.83%
10	9	v	Perl	2.032%	-0.04%

Objective-C est plus que jamais en dehors du top 10. Swift devrait désormais très rapidement le dépasser et intégrer au printemps 2016, les dix premiers. Pour le reste, peu de changement : Java, C et C++... Python, sur lequel nous avons fait récemment un focus, est très en forme (5e).

agenda

Belgique

Dev Day : 1er décembre (Mons, Belgique)

L'évènement développeur belge se tiendra le 1er décembre à Mons. L'évènement sera très orienté sur les technologies Microsoft et Windows 10 mais aussi Raspberry Pi (+ Windows 10). A cette occasion des speakers de renom seront présents : David Rousset, Sébastien Warin, Samuel Blanchard... Pour les étudiants, l'évènement est gratuit.
Pour en savoir plus : http://www.devday.be

Les événements Zenika

Matinale Agile Wake Up / 01 décembre (Zenika Paris)
Pour ce lancement, nous vous proposons de vous initier à l'Agilité et de vous montrer qu'elle peut s'adapter à différents contextes à travers quatre conférences.
Programme et inscription :
http://www.zenika.com/agile-wake-up-1.html
Zenika organise le NG2 Tour au sein de ses différentes agences et vous propose de découvrir tous les secrets sur Angular 2. Vous souhaitez commencer une nouvelle application et vous hésitez entre AngularJS et
Angular 2 ? Vous désirez avoir un aperçu de la prochaine version du framework ? Ou peut-être faites-vous partie des personnes qui ont des interrogations par rapport à ce qu'ils ont déjà entendu sur l'avenir d'Angular ?
Les dates :
Zenika Rennes : 03 décembre
Zenika Nantes : 04 décembre
Zenika Paris : 12 janvier 2016
Zenika Lyon : 21 janvier 2016
Zenika Lille : 26 janvier 2016
Programme et inscription :
http://www.zenika.com/Evenements/zenika-ng2-tour.html

Mobile Dev 10 décembre

L'école ESGI organise une nouvelle édition de sa journée développement mobile, sur le campus de Paris. La journée sera ponctuée de conférences et de rencontres.

COMMANDEZ WINDEV 21

OU WEBDEV 21 OU WINDEV MOBILE 21

ET RECEVEZ UN SUPERBE MATÉRIEL

Prolongation exceptionnelle jusqu'au 18 Décembre

4K - 140cm

TV SAMSUNG

140cm 4K
3.840 x 2.160
Wi-Fi

Réf: UE55JU6000

Ou choisissez :

- TV HD 3D Wi-Fi 140 cm
  Réf UE55H6400

ORDINATEUR DELL

17'' Core i7 tactile 2To 12Go

Ou choisissez parmi:

- Station de travail **DELL Précision T1700** Mini-tour Disque 500Go Mémoire 8Go Windows 7 Pro + 8.1 Pro (MAJ 10 incluse)
- «All in One» PC **DELL 23" Inspiron 5348** Windows 8.1 (MAJ 10 incluse) Intel Core i5 Disque 1 To - tactile

D'autres matériels sont proposés sur le site www.pcsoft.fr

Aucun abonnement à souscrire. Compatible tous opérateurs

SAMSUNG GALAXY S6 EDGE

5,1'' 64Go
Bords incurvés
Android

Ou choisissez parmi:

- 1x **Galaxy S6 Edge+** 32 Go
- 2x **Galaxy S5 New**
- 2x Tablettes **Galaxy Tab 2S** 9,7"

OPÉRATION POUR 1 EURO DE PLUS

Pour bénéficier de cette offre exceptionnelle, il suffit de commander WINDEV 21 (ou WINDEV Mobile 21, ou WEBDEV 21) chez PC SOFT au tarif catalogue avant le 18 Décembre 2015: pour 1 Euro de plus, vous recevrez alors le ou les magnifiques matériels que vous aurez choisis. Offre réservée aux sociétés, administrations, mairies, GIE et professions libérales, en France métropolitaine. **L'offre s'applique sur le tarif catalogue uniquement**. Voir tous les détails et des vidéos sur : www.pcsoft.fr ou appelez-nous (04.67.032.032).
Le Logiciel et le matériel peuvent être acquis séparément. Tarif du logiciel au prix catalogue de 1.650 Euros HT (1.973,40 TTC). Merci de vous connecter au site www.pcsoft.fr pour consulter la liste des prix des matériels et les dates de disponibilité. Tarifs modifiables sans préavis.

www.pcsoft.fr

Fournisseur Officiel de la Préparation Olympique

Elu
«Langage le plus productif du marché»

Choisir son école d'informatique/de développement

Depuis plusieurs années, nous vous parlons régulièrement des écoles, et de nombreux numéros spéciaux sortent sur ce thème dans la presse. Une des questions qui revient régulièrement : comment choisir ? Mais cette question se révèle rapidement complexe : quels critères ? Quel type d'école et de cursus (et de diplôme donc) ? Programmez ! apporte quelques réponses à vos angoisses.

François Tonic
Programmez!

Que vous soyez encore au lycée, avec ou sans bac, déjà étudiant ou même éventuellement déjà (jeune) développeur, il existe des écoles et des cursus faits pour vous, même en France. La culture du diplôme demeure très importante, au risque de décourager des jeunes à se lancer dans l'aventure de l'informatique, du développement, du numérique. Même si les mathématiques ne vous passionnent pas, que vous ayez un bac pro ou arrêté les études avant le bac, vous pouvez tout de même faire carrière dans le développement, l'informatique. Il est certain qu'à court terme les métiers du numérique au sens large vont continuer à croître et il y aura pour plusieurs années des pénuries, tout le moins, un manque de compétences, notamment dans le développement. *« La pénurie ne va pas décroître. À notre dernier forum, nous avions 7000 propositions (de stages) pour 2000 élèves. Et 20 % partent à l'étranger. »* Précise **Joël Courtois** (directeur général EPITA).

Cependant, il ne faut pas croire que tout le monde deviendra « informaticien », ou développeur. Il y a sans aucun doute parmi les élèves en échec, des pépites qui méritent une chance et d'être aidé.

Une grande diversité d'écoles et de filières

« Il existe beaucoup d'écoles, beaucoup de modèles différents et une grande diversité, telle que l'école 42. » précise Emmanuel Peter (directeur ESGI). Les filières principales sont :

Filière/école	Durée
BTS	2 ans après le bac
IUT	bac+2/+3
Université	jusqu'au doctorat et post-doc
École d'informatique	de bac+2 à bac+5
Alternance	en général 2 ans, mais aussi en bac+5
École d'ingénieur	bac +5
École de type 42	3 ans

Toutes les formations n'exigent pas un bac ou un diplôme comme préalable. En école d'ingénieur, en général, on accède après une prépa. Pour plus d'informations, consultez le tableau général disponible sur www.programmez.com. Selon la filière, la formation et le diplôme ne seront pas identiques, que ce soit dans la formation en elle-même des fondamentaux du développement et de la technique, ou dans la pratique (stages, projets).

Un choix multiple, mais finalement un seul choix

Les premiers critères de choix sont très simples :

- Si vous avez le bac ou non : si oui, le type de bac peut éventuellement jouer;
- Selon les prérequis nécessaires pour telle ou telle filière. Par exemple, pour l'université, une école d'ingénieur, une école informatique, le bac est nécessaire. Mais le bac n'est pas l'unique condition. La mention, le type de bac (scientifique, pro, littéraire...) seront bloquants, ou du moins, handicapants pour certaines filières. À cela peut se rajouter une prépa. Pour les grandes écoles, écoles d'ingénieur, la prépa est souvent conseillée, mais chaque école a son mode de sélection.

> *« On parle beaucoup des décrocheurs. Il y a des talents à découvrir »* (Aston)

Un élément que l'on oublie souvent : faire un travail sur soi-même et se demander réellement la carrière que l'on veut faire : quels profils m'intéressent le plus, suis-je motivé pour le développement, la technologie ?

On utilise le terme très général d'informaticien. Mais aujourd'hui informaticien ne signifie plus grand-chose, car il existe des dizaines de profils très différents (technique, non-technique, développeur, architecte logiciel, marketing web, design, etc.). Si vous souhaitez être rapidement opérationnel et que vous n'aimez pas les études, mieux vaut choisir une filière courte, des écoles de type 42. Il sera ensuite possible de compléter votre formation initiale par une formation ultérieure ou des certifications pour monter en compétence. Parfois il y a un choc de culture entre le lycée et l'école supérieure. *« Au début oui c'est un peu le choc, mais nous sommes très bien encadrés. Des copains en IUT ou en fac le sont bien moins. »* précise **Bastien Robert** (L2 promotion 2019, EFREI).

Votre niveau scolaire et votre Bac (mention, type de bac), comme nous l'avons dit plus haut, seront déterminants pour certaines écoles (ex. : écoles d'ingénieurs) nécessitant souvent un bac scientifique et une mention. Si votre rêve est d'intégrer une école d'ingénieurs, il est parfois possible d'être admis par des voies secondaires.

Certaines écoles misent aussi sur la flexibilité des programmes et des technologies même si les fondamentaux changent peu. *« Parfois les écoles sont lentes à se mettre à jour. Nous, on peut changer nos technologies tous les 6 mois si besoin. »,* nous indique Corinne Combes (directrice Aston). Sans être aussi rapide, il est important de bien comprendre les formations, les cours, les stages proposés et les technologies/langages que l'école propose. Dans le monde Web, un apprentissage du Flash n'aura plus d'intérêt par exemple ni du HTML avant la v5.

Bac+5 : ingénieur, niveau ingénieur

Il faut être honnête, il existe une certaine obsession du diplôme d'ingénieur, le fameux bac+5. Beaucoup d'entreprises demandent, exigent ce diplôme et les annonces l'indiquent souvent. Mais d'excellents développeurs ou sysadmin n'ont pas ce sésame et font de belles carrières. Un Bac+5, c'est 5 ans d'études après le bac. C'est long et les rythmes exigés sont élevés. Il faut tenir et être capable de financer 5 ans d'études. Aujourd'hui, environ 210 écoles délivrent le diplôme d'ingénieur. Ces écoles sont publiques ou privées (beaucoup sont privées).

Il existe plusieurs filières bac+5 :

- Les écoles d'ingénieurs, par exemple les écoles des mines. Ce sont des écoles dites généralistes avec une forte dominante scientifique.
- Les écoles bac+5, école d'ingénieur en infor-

Vous croyez **encore que le** *Cloud Computing* **est cher ?**

Avec ArubaCloud,

Vous avez accès à une large gamme de solutions de Cloud Computing, avec choix du pays du datacenter utilisé, solution packagée ou flexible avec facturation adaptée..
Vous pouvez dès maintenant créer votre serveur Cloud SMART à partir de 1€ht / mois.

*Prix Hors Taxe, vérifiez la liste des prix pour plus d'informations

**MON PAYS, MON CLOUD.

Hyperviseur vmware

Contrôle des coûts

6 datacenters en Europe

APIs et connecteurs

Linux & Windows

1 Quitte à choisir une infrastructure IaaS, autant prendre la plus performante!
Aruba Cloud est de nouveau **N°1 du classement des Cloud**
JDN / CloudScreener / Cedexis (Janvier 2015)

Contactez-nous! 0810 710 300 www.arubacloud.fr

aruba CLOUD

Cloud Public | Cloud Privé | Cloud Hybride | Cloud Storage | Infogérance

MY COUNTRY. MY CLOUD.**

matique. Elles ne délivrent pas forcément un diplôme d'ingénieur, mais de niveau bac+5. Par exemple l'école Etna qui fournit un titre bac+5 en alternance.

Il existe différents bac+5 et cursus d'ingénieurs. Vous pouvez opter pour une école d'ingénieur généraliste surtout si vous êtes bon en mathématiques et en sciences. *« Les Mines Nancy n'est pas une école informatique, mais une école d'ingénieur. Nous avons des parcours très variés et différents départements. Les étudiants ne vont pas faire 30 heures de C++, 30 de Java, etc. Nous sommes une école généraliste avec un bon niveau scientifique, mais nous regardons aussi les technologies comme le Big Data. Nous proposons aussi des spécialisations via plusieurs masters en informatique, sécurité informatique. »* recadre **Bart Lamiroy** (Mines Nancy)


© Inria

Les écoles d'informatique proposent souvent différents domaines de compétences pour leurs formations et diplômes. Par exemple en bac+5 : Sup'Internet se concentrera sur le Web (développement, marketing, design), Hetic sur le Web et la 3D, ESILV se concentrera sur le big data, web, mobile, embarqué, EFREI aura une approche large (architecture SI, BI, sécurité, imagerie, développement).

Ne pas sous-estimer le coût des études

Comme nous l'avons dit plus haut, le bac+5, quelle que soit l'école choisie, est exigeant, long et contraignant. Vous devez peser le pour et le contre immédiatement. Et vous devez être conscient(e) du coût financier en école privée. Une solution en alternance permet de mieux supporter les frais scolaires, même si toutes les formations en alternance ne permettent pas d'être en entreprise dès la 1ere ou 2e année.

La plupart des écoles que nous avons interrogées durant cette enquête ont des partenariats avec des banques pour faciliter les prêts étudiants, voire, parfois, avec des résidences ou des organismes spécialisés pour aider l'étudiant à se loger. Le logement est parfois un point délicat (et qui pèse lourdement dans le budget annuel). De nombreuses écoles aménagent les frais de scolarités en cas de difficultés. Certaines écoles sont plus proactives que d'autres sur ces sujets.

Tous les étudiants que nous avons rencontrés se sont posé la question à un moment donné. *« Oui j'appréhendais un peu. Mes parents m'ont aidé et ont débloqué l'argent. Parfois, on en discute entre étudiants et se pose aussi la question de l'argent qui reste pour vivre, sortir. »* précise Thomas (1ere année Epitech). L'étudiant habite chez ses parents ou loue ou co-loue un appartement.

Le financement doit être pris très au sérieux quand vous optez pour une école privée, quel que soit le cursus, court ou long. *« Que faire ? BTS ? IUT ? C'est bête de se fermer des portes. Qu'est-ce que cela m'apporte ? J'ai calculé sur 5, 10, 15, 20 ans. J'ai regardé les statistiques des salaires. J'ai cherché un logement, une banque. »* évoque Bastien Robert.

Il y a aussi celles et ceux qui ne souhaitent pas prendre un prêt étudiant ni demander aux parents. C'est notamment le cas de Sarah Hathat (étudiante 2e année, Supinfo), qui travaille le week-end pour financer l'école. *« C'est une contrainte, mais c'est un choix que j'ai fait même si c'est parfois un peu dur. Cette année je n'ai pas cours le jeudi cela me permet de sortir un peu ou de travailler mes cours. J'aimerais bien un contrat de professionnalisation l'an prochain pour payer l'école (l'alternance se fait à partir de la 3e année à Supinfo, NDLR). »* précise **Sarah Hathat**.

Ne pas hésiter à changer de filières, à vous remettre en question ou à plonger

L'informatique et le développement peuvent intéresser tout le monde ! Et comme nous l'avons vu, les filières sont nombreuses. Et vous pouvez parfaitement récupérer une formation informatique/développeur tout de suite après le bac ou plus tard. *« J'ai fait une année d'université. J'ai voulu changer. J'ai regardé sur Internet et je suis allé à des salons étudiants. J'ai pu y voir des stands d'écoles, dont Supinfo. Cela m'a plu. Je n'avais jamais fait de technique ni de développement. J'avais un peu peur au départ et je me suis lancée. »* Raconte Sarah Hathat.

Il est difficile de savoir ce que l'on va faire comme métier. Et quand on termine le lycée, il faut bien choisir quelque chose. Aujourd'hui, il est plus facile de changer de filières. Il ne faut pas hésiter à se remettre en question et tenter.

On peut hésiter entre plusieurs filières. Par exemple **Florian Toix** avait hésité entre l'informatique et la biologie. Il a finalement opté pour l'informatique en discutant avec un ami qui était déjà à l'EPSI.

« Un ami m'avait parlé de l'Epitech, où il était étudiant. J'ai commencé à regarder ce que proposait l'école et j'ai fait plusieurs visites. Je trouvais cela sympa et je me suis dit pourquoi pas. J'aimais bien l'informatique, mais pas forcément pour en faire (un métier). L'Epitech était mon choix, l'école où je voulais aller. » précise Thomas.

« Dès le collège, je suivais les cours d'Openclassroom. Le fondateur sortait de l'EFREI. Je me posais des questions sur l'après bac. J'ai fait des recherches, je suis allé aux forums, aux portes ouvertes. J'ai discuté avec des anciens, des étudiants » raconte Bastien Robert. Le choix de l'école dépend beaucoup de vous. Il faut chercher, aller aux salons

Non, il n'y a pas que le bac+5 dans la vie !

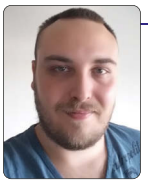
Soyons honnêtes, aujourd'hui, les entreprises demandent un niveau bac+5, à tort ou à raison. Cependant, il ne faut pas que cela soit une obsession au risque de vous décourager et de vous faire abandonner les technologies et l'informatique. Malgré tout, il faut être conscient des problèmes potentiels que pose le niveau de formation auprès des entreprises. « Je n'étais pas sûr d'être embauché à bac+2 et avec mon BTS. Je me suis posé la question sur le bac+5. J'ai vu l'opportunité du titre d'ingénieur, mais je n'avais pas les finances pour le faire. Une école privée a un coût. J'ai cherché une école avec de l'alternance. J'ai eu la chance de trouver l'école et une entreprise. » précise Florian Toix (ingénieur étude et développement, diplômé EPSI Bordeaux).

Durant notre enquête, une des craintes soulevées était l'embauche immédiate avec un BTS ou toute autre formation courte à bac+2/3 et parfois aussi, les possibilités d'évolution de carrière et le niveau de salaire. Mais ne soyez pas non plus obnubilé par le diplôme et le bac+5. Car un bon développeur doit être agile, à la fois autonome et communicant et maîtriser les technologies et les langages. Prouvez vos compétences au quotidien, et n'hésitez jamais à investir du temps sur la veille technologique et à vous former constamment à de nouveaux langages et frameworks.

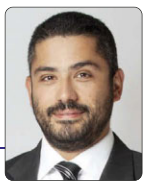
étudiants, venir aux portes ouvertes pour voir les locaux, l'ambiance, discuter avec les étudiants, les profs, consulter les cursus et formations. Finalement, les orientations au lycée évoquent encore peu les écoles d'informatique et les nombreux métiers du numérique. Vous devez être volontaire et ne jamais hésiter à aller voir par vous-même.

Un peu de 42 et d'alternance

L'école 42 est atypique dans le paysage. On peut y entrer sans diplôme et c'est ouvert à tout le monde (avec limite d'âge tout de même). La sélection n'en demeure pas moins très stricte et sévère : des tests en ligne puis la fameuse « piscine », une immersion totale durant plus d'un mois, en groupe. Même si effectivement, on n'a pas besoin d'être codeur dans l'âme, si vous arrivez à la piscine, vous allez très vite devoir apprendre, coder, travailler en équipe pour décrocher une place parmi le millier de sélectionnés chaque année. La formation dure 3 ans et très orientée pratique, projets. Pas de diplôme reconnu à la sortie, mais assurément une pratique du code. Mais ce modèle ne convient pas à tout le monde.



Sébastien, 24 ans, a déjà vécu plusieurs vies (brancardier, dans la pâtisserie) mais l'informatique l'a toujours passionné. *« Un ami m'a parlé de 42 qu'il voulait intégrer. Je ne connaissais pas du tout, il m'a alors expliqué le principe. Et j'ai voulu tenté même si je n'y connaissais rien à la programmation mais je sentais que c'était ce que je voulais faire »* explique Sébastien. Il a passé avec succès les tests de logiques puis le staff l'a contacté. Puis l'épreuve redoutée de la piscine était arrivée. *« On se fait beaucoup d'idées sur la piscine. Je pensais que je n'y arriverais jamais. J'ai alors regardé des vidéos, discuter sur les forums. J'étais un peu en panique. Mais dès les premiers jours je n'ai pas trouvé cela aussi terrible même si elle est exigeante. »* poursuit-il. La piscine demande beaucoup de travail et de rigueur. L'entraide entre les élèves est très importante pour s'aider mutuellement et avancer. Même si le langage C ne parle pas totalement à Sébastien, il a fait le job. Les premières semaines sont intenses, +10 heures par jour... Sébastien va vivre fin novembre l'entrée en 1ere année même s'il ne sait pas encore ce qu'il va faire exactement, peut-être plus vers le web. La motivation et la volonté de réussir sont là ! Sans faire du 42 à 100 %, aujourd'hui, les écoles d'informatique imposent de nombreux projets dans l'école ou des stages ou de l'alternance en entreprise, parfois dès la 1ere année. *« L'étudiant est mis en situation grâce à l'alternance. Il applique ce qu'il apprend à l'école. Mais il est vrai que l'alternance n'a pas toujours bonne image. »* précise **Samir Rinaz** (Etna).



À vous de vous vendre !

Nous avons déjà évoqué à plusieurs reprises la motivation et des éléments qui peuvent plaire aux écoles. Il n'y a pas de profils types des nouveaux entrants, car selon l'école, celui-ci change beaucoup. L'école Aston évoque un âge moyen relativement élevé 27/28 ans. Il n'est pas introverti et il sait se vendre, communiquer. Un bon équilibre de vie est parfois encouragé (sortie, détente, sport...).

Ne pas négliger l'Anglais

Ce n'est pas une surprise : l'Anglais est la langue du numérique et du développement. Beaucoup de ressources sont uniquement en Anglais. Et de nombreuses écoles (ingénieur/informatique) proposent des cours en Anglais. On ne vous demande pas d'être bilingue, mais de savoir vous débrouiller, de soutenir un projet et d'échanger. Certains sont plus à l'aise que d'autres. « *Mon niveau d'Anglais n'est pas mauvais, mais le niveau (à l'école) est exigeant. Il faut taffer !* » indique Thomas.

« *Globalement, le niveau est moyen.* » Précise **Emmanuel Carli** (directeur général Epitech). Epitech utilise, comme d'autres écoles, les grilles d'évaluation. Et, par exemple, l'Epitech impose une année à l'international (4e année). Et c'est obligatoire.

Là encore, il ne faut pas avoir peur de se lancer et d'apprendre même si vous n'êtes pas bon. Je connais des développeurs qui sont partis aux USA avec un niveau moyen/médiocre, mais ils ont saisi les opportunités proposées...

Des promos avec plus de filles !

Autre constat, les filles sont toujours peu nombreuses dans les filières informatiques. Certaines affichent de bons pourcentages. « *Nous avons au moins 20 % de filles. Dans la dernière promo, nous atteignons 22 %. Avant, elles s'orientaient beaucoup vers le design/marketing. Cette année, par exemple, 20 % d'entre elles choisissent l'option technique.* » Constate **Isabelle Clary** (directrice de Sup'Internet). Nous avons posé la question à Sarah : « *Cela ne m'a pas fait peur qu'il n'y ait pas beaucoup de filles. C'est vrai que parfois c'est un peu lourd, mais l'ambiance est bonne.* »

Pour conclure

La réussite dépendra de votre motivation. La motivation est souvent prise en compte dans les entretiens d'admission. Et l'univers du numérique est vaste avec des dizaines de métiers très différents. Il n'y a pas que le développement ou la technique pure. Et tout dépendra aussi de vos envies et ambitions dans le métier : trouver un job très rapidement, faire une vraie carrière dans le développement, etc. La France possède de très nombreuses écoles (ingénieur/informatique) et les différentes filières sont en général accessibles dans votre région. Il n'y a pas que Paris et sa région... Ainsi à Bordeaux, une nouvelle école ouvrira en septembre 2016 : ESN / ESD. La formation sera de 3 ans après le bac (le bac est obligatoire) et des masters. La pédagogie sera assez classique.

Nous remercions toutes les écoles et les étudiants qui ont répondu à notre enquête. Nous n'avons pas pu citer tout le monde. ⊞

DES FORMATIONS ET CURSUS SANS PRÉREQUIS OU SE FORMER RAPIDEMENT

Dans ce dossier, nous parlons beaucoup d'écoles, de cursus après le bac, etc. Mais il n'y a pas que ces filières plus ou moins longues. Vous avez des structures telles que Simplon.co. Il s'agit d'un réseau dans toute la France pour former en quelques mois des jeunes et moins jeunes aux technologies du Web, à la programmation mobile.

Simplon se destine plutôt aux – 20 ans, avec ou sans diplômes. Ces formations sont intensives (6 mois) et très denses, basées sur la pratique et les projets. Si les portes sont ouvertes à tout le monde, la motivation est un des critères essentiels. À la clé, on peut espérer devenir intégrateur web ou développeur junior. Cette formation est une aide et une base, mais il faut ensuite consolider et étendre ses compétences.

Un peu dans la même veine, le pôle Leonard de Vinci (Nanterre) lancera début 2016 la « DevSchool ». « Nous avons des velléités d'aider les personnes en difficulté et proposer une autre façon d'apprendre, notamment avec l'initiative Grande École du Numérique. » évoque

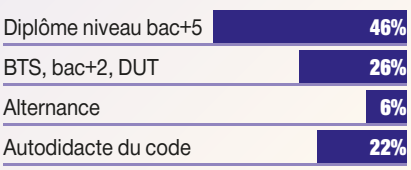
Jérôme Da Rugna (directeur de la formation ESILV). L'objectif est de proposer des programmes de formation de 6 mois. La DevSchool va au-delà avec des cursus de 2 ans avec ou sans le bac avec de la programmation, du développement mobile, de la gestion de projet et de la culture digitale.

Récemment, le gouvernement a lancé l'initiative « Grande Ecole du Numérique » dont les objets sont simples : former aux métiers du numérique 10 000 jeunes sans emploi et sans qualifications d'ici 2017 (délai un peu court, mais bon). Cette initiative doit s'appuyer sur 200 fabriques numériques. L'ambition métier est importante, car elle concerne le développement, l'infographie, le webmarketing, le réseau, la gestion des communautés, le webdesign et la maintenance/support informatique.

VOUS ET VOS ÉTUDES

Notre sondage express montre que vous êtes 46 % à avoir une formation bac+5. Ce qui est intéressant, ce sont les 22 % d'autodidacte du code qui n'a pas forcément un diplôme en informatique. Ce résultat nous a un peu surpris. Pour le reste, vous êtes 6 % en alternance et 26 % ayant fait un bac+2, DUT, BTS.

Nombre de votants : 338



Check-list

Impossible de proposer une liste de conseils valable pour tout le monde. Voici quelques pistes et idées pour vous aider : n'hésitez pas à vous renseigner avant le bac sur les filières, les écoles, les carrières possibles;

- **Quels métiers m'intéressent ? Répondre à cette question permettra de faire une liste de filières et d'écoles;**
- **Est-ce que je suis prêt(e) à des études longues ? Ce choix est important pour choisir la filière et le type de formation. Cela évitera des perdre des années et de vous décourager. Même si le taux d'échec dans les écoles d'informatique est faible il existe;**
- **Si vous avez des amis dans une école d'informatique ou de la famille travaillant dans l'informatique, posez des questions;**
- **Allez aux portes ouvertes, dans les forums, cherchez et informez-vous ! Les collèges et lycées sont encore trop timides sur le numérique;**
- **Ne négligez pas la partie financière. Une école privée coûte cher et n'oubliez pas d'intégrer le logement, les transports et tous les à-côtés (loisirs, sports, assurances, matériels, etc.)**

À vous de jouer maintenant.

DOMAINES | MAIL | HÉBERGEMENT | E-COMMERCE | **SERVEURS**

TESTEZ LE MEILLEUR CLOUD

TESTÉ ET APPROUVÉ PAR CLOUD SPECTATOR

Powered by intel Cloud Technology

1&1 Serveur Cloud : Easy to use – ready to Cloud*

Les performances des nouveaux serveurs Cloud sont imbattables en termes de CPU, RAM et SSD.

Réalisez vos projets dans le Cloud avec la combinaison parfaite entre flexibilité et performances.

- ✓ **Load balancing**
- ✓ **Stockage SSD**
- ✓ **Facturation à la minute**
- ✓ **Intel® Xeon® Processor E5-2660 v2 et E5-2683 v3**

1 mois gratuit !

Puis à partir de 4,99 € HT/mois (5,99 € TTC)*

1 MOIS POUR ESSAYER

1 CLIC POUR CHANGER DE PACK

1 APPEL UN EXPERT VOUS RÉPOND

0970 808 911
(appel non surtaxé)

1&1

1and1.fr

*Facile à utiliser - prêt pour le Cloud. 1&1 Serveur Cloud : 1 mois d'essai gratuit, puis à partir de 4,99 € HT/mois (pour la configuration du serveur Cloud S). Facturation mensuelle en fonction de la configuration choisie. Pas de durée minimum d'engagement, ni de frais de mise en service. Conditions détaillées sur 1and1.fr. Intel et le logo Intel sont des marques commerciales d'Intel Corporation aux États-Unis et/ou dans d'autres pays. 1&1 Internet SARL, RCS Sarreguemines B 431 303 775.

Ecole, diplôme, travail

« Tout ce que nous avons à décider, c'est ce que nous devons faire du temps qui nous est imparti. »

Gandalf.

Frédéric Fadel
Mathématicien de formation, autodidacte en informatique, cofondateur de Winwise (1993-2008) une société de service en informatique, cofondateur d'Aspectize (2007) un éditeur de logiciel qui métamorphose l'expérience de développement d'applications Web et Mobile.

Pourquoi un diplôme ?

Il y a le passé, ses habitudes, ses croyances, ses craintes, ses dysfonctionnements, ses métiers, son incapacité à imaginer le monde autrement, sa mentalité.

Parmi les habitudes, il y a l'école : on va à l'école par habitude c'est-à-dire qu'on le fait spontanément, sans se poser de questions, comme quelque chose qui doit être fait, quelque chose de normal.

On ne se pose pas la question « pourquoi va-t-on à l'école ? », parce que la réponse évidente est qu'on veut un diplôme, et les questions se transforment en : quel diplôme ? De quelle école ? En combien de temps, 2 ans, 3 ans, 5 ans ou plus ? Quel est le prestige du diplôme, quelle est la supériorité de l'école qui le délivre ? On ne se pose pas la question « pourquoi veut-on un diplôme ? », parce que la réponse flagrante est qu'on veut du travail, et les questions se transforment en : quel employeur ? Quel salaire ? Combien d'heures de travail par semaine ? Quelles perspectives d'évolution ? Quand est-ce qu'on devient chef ? Il y a de vieux métiers, de vieilles matières. Si on veut être médecin ou mathématicien, il faut composer avec des milliers d'années d'histoire et de façons de faire, il faut pouvoir prouver qu'on est digne des anciens en obtenant un diplôme délivré par le jugement des autres, une sorte de label AOC. Mais il y a aussi de nouveaux métiers, de nouvelles façons de faire. L'informatique aujourd'hui est un domaine naissant. On a la chance de pouvoir contribuer, chacun à sa manière, à sa création, sa pratique et à son développement.

Nous n'en sommes qu'aux débuts de l'informatique

L'informatique n'en est qu'à ses débuts, tout est encore à imaginer. Pour le moment nous avons numérisé la façon de travailler du passé. 80 ans après la machine de Turing, 40 ans après le microprocesseur et 20 ans après le Web, nous continuons d'utiliser silicium et fibres pour pourvoir à l'imagination ancienne. Même en utilisant des programmes, on fait de la comptabilité et du classement à l'ancienne. On a remplacé la machine à écrire par le traitement de texte mais on écrit les mêmes textes. On continue à travailler pour un diplôme ou un salaire comme nos parents le faisaient au XXe siècle. *L'informatique c'est l'art de faire circuler l'information entre les machines et les hommes*. Les machines, ordinateurs, téléphones, détecteurs, récepteurs et émetteurs sont partout et leur prolifération ne fait que commencer.

L'informatique du XXIe siècle, consiste à inventer de nouveaux usages, imaginer de nouvelles façons d'utiliser les câbles et les octets.

Avec ses moyens de communication monodirectionnels - livre, presse, radio, télé -, le passé nous a contraints à adopter une posture de diplômé soumis.

Grâce à l'informatique et à la communication réticulaire, nous adopterons - dans l'avenir - la posture de celui qui écrit et qui parle, celui qui fait et agit, depuis partout et à n'importe quel moment. Nous serons celui qui facilite, permet et connecte, nous serons les inventeurs épanouis du nouveau monde.

Quand Marc Andreessen explique que « Software is eating the world », il est en train de parler de l'émergence de ce monde nouveau, libéré des contraintes matérielles imposées par l'industrie, l'école, la centralisation et la hiérarchisation. Un monde où les possibilités et les valeurs se créent grâce aux logiciels et aux échanges entre individus facilités par des machines. Ce nouveau monde n'est pas un monde de consommateurs passifs mais un monde de producteurs actifs qui s'échangent leurs services, leurs savoirs, leurs compétences.

Qu'est-ce qu'on apprend à l'école ?

Qu'est-ce qui peut être enseigné ?

Certainement pas l'imagination, la créativité, l'inventivité. Au mieux, vous aurez un prof compétent et un environnement sympa pour apprendre l'informatique du passé, celle qu'on connait déjà, celle qui simule le fonctionnement du passé avec les processeurs et les disques durs, celle qui exploite les vieilles idées.

Vous pourrez apprendre un ou plusieurs langages de programmation, vous serez asséné avec la mythologie de l'informatique, des algorithmes, de l'orienté-objet, des « design patterns », de l'UML, du SQL, de TCP/IP, et de NoSql pour faire moderne.

Vous aurez peut-être un cours sur l'agilité, mais il sera délivré magistralement, à une heure donnée, dans une salle donnée ; vous serez jugé, noté et éventuellement diplômé sur l'agilité mais vous ne l'aurez pas pratiquée. La bureaucratie, les feuilles de présence, les excuses d'absence, les devoirs et les dates de remises seront sur le chemin de votre apprentissage. Ce processus est tout sauf agile et il vous faudra beaucoup de confiance en vous pour ne pas céder et ne pas vous faire imprégner par cette éducation implicite. Education qui vous prépare pour le monde du passé.

Vous n'apprendrez pas à développer et imaginer les logiciels de demain, ceux qui feraient de vous un acteur du nouveau monde.

Cela, vous le ferez *malgré* vos diplômes, vous le ferez d'autant plus facilement que vous ne serez pas abimé par vos diplômes et les besoins du passé, vous le ferez parce que vous aurez hâte de créer le futur, parce que ça vous amusera.

Pour préserver la motivation, pour préserver la spontanéité, pour préserver la confiance et le fun, Peter Thiel a lancé en 2011 un programme baptisé « 20 under 20 » (http://thielfellowship.org). Il sélectionne sur dossier 20 jeunes de moins de 20 ans dotés d'un talent exceptionnel pour les nouvelles technologies, et leur donne à chacun 100 000 dollars sur deux ans pour les aider à monter une start-up, une ONG ou un projet de recherche. A une condition : qu'ils abandonnent leurs études.

Nos amis Gates, Jobs, Zuckerberg, Dell, Allen, Dorsey, Williams, Koum, Niel ou encore Zola, Guitry, Malraux, Brassens ont réussi sans diplôme. D'autres comme Bezos, Kalanick, Chesky, Page, Brin ont réussi avec diplôme mais pas grâce à leur diplôme.

On peut croire que ces célébrités sont des exceptions. Mais ce qui est exceptionnel, c'est leur réussite. Il y a beaucoup d'autodidactes, passionnés et compétents qui réussissent leur vie sans faire de bruit. Certes, aujourd'hui et particulièrement en France, le pédigrée scolaire sert encore à sélectionner, trier et classer les futurs *employés* - ce terme ne vous embarrasse-t-il pas ? -, mais plus pour longtemps. Les temps changent, les métiers changent, les relations au travail changent, il est donc temps de

Mission: MAKER

Un pack pour bien démarrer avec Arduino

Programmez ! vous propose un kit de démarrage comprenant une carte Arduino Nano et les composants essentiels (écran LCD à souder, LED, résistances, moteur, planche à pain, etc.).

SEULEMENT 30 PACKS DISPONIBLES !

35€
+ 4€ de frais de logistique*

CE PACK COMPREND :

- 1 carte compatible Arduino Nano (version CH340 driver), avec câble USB**
- 1 écran LCD (headers non soudés)
- 1 photorésistances
- 1 capteur de température DS18B20
- 3 boutons poussoirs
- 15 LED
- 18 fils
- 1 planche à pain
- 1 capteur HC-SR04
- 1 buzzer
- 1 moteur SG-90
- 1 lot de résistances (différentes valeurs)
- 1 lot de headers
- 2 LED infrarouge (1 émetteur / 1 récepteur)
- 1 clip pour pile 9V

Commandez dès aujourd'hui sur le site internet : www.programmez.com

** Le pack est disponible uniquement pour la France.*

** Cette version n'est pas compatible avec OS X. Testé avec Arduino IDE et Visual Micro (à installer sur Visual Studio) sur Windows 10.
Pas de documentation incluse. Vous trouverez facilement des tutoriels de démarrage sur le web. Plusieurs livres Arduino disponibles chez ENI, Eyrolles, Dunod. Des tutos seront proposés sur www.programmez.com courant décembre 2015.
Les expéditions se feront à partir du **1er décembre 2015**.

se préparer à changer de mentalité et de contribuer au futur, et ça c'est une affaire privée, aucune école ne vous l'apprendra.

Accessibilité des outils et de l'information

Par ironie du sort, un des domaines les plus fortement minés par les nouvelles manières de faire, c'est le domaine de l'éducation.
La technologie, les MOOC, les Meetup, les ressources en ligne, les moteurs de recherche, les communautés de techniciens compétents comme stackoverflow, les conférences techniques en ligne, le Cloud, les outils de développements gratuits, et même les cours classiques des universités du monde entier... Ce dont on a besoin pour apprendre, pratiquer et s'améliorer en informatique est disponible en ligne.
Avec l'informatique, nous pratiquons une activité d'avenir, une activité qui dans un certain sens n'existe même pas clairement aujourd'hui. C'est une activité de créateurs, une activité pour ceux qui se projettent dans l'inconnu pour imaginer le monde de demain.
L'informatique est en train de changer la société, comme les routes et l'industrie l'ont fait à leur époque. Sauf qu'au lieu de contraindre et séparer, elle libère et connecte, au lieu de favoriser la compétition et la propriété, elle favorise la collaboration et le partage. La plupart des métiers de demain n'existent pas aujourd'hui, aucun des diplômes d'aujourd'hui n'aura de valeur demain.

Marché du travail

Quand on cherche en ligne « travailler sans diplômes », à part les annonces de formations, on tombe sur des articles qui parlent de « travailler sans compétences » ou « travailler sans qualification ». Cela montre que le marché du travail est encore assez passéiste, du moins en France. Néanmoins, comme je l'ai souligné ici, les temps sont en train de changer : le bouillonnement des startups, la prolifération des hackathons, les sites pour freelances présentent des opportunités nouvelles pour réussir sans diplôme en autodidacte. Etre autodidacte veut dire que vous construirez vos compétences selon vos envies, selon les opportunités que vous rencontrerez, selon le rythme qui vous conviendra. La meilleure façon d'apprendre c'est de faire, et il y a un marché pour la réalisation informatique en freelance pour tous les niveaux de compétences. La question à poser c'est : voulez- vous passer 2, 3, 5 ans de votre vie dans une école pour décrocher un diplôme avant d'être autorisé à faire, ou vous voulez commencer à apprendre en faisant ?
Dans le premier cas vous aurez un diplôme, peu d'expérience et vous aurez subi les contraintes de l'école. Dans le second cas vous n'aurez pas de diplôme, vous aurez plus d'expérience, vous aurez suivi vos envies et probablement vous aurez gagné un peu d'argent en faisant.
La mentalité du passé, c'est de vivre avec les habitudes et les craintes du passé, c'est penser que demain sera comme hier, et répéter les cérémonies et incantations, en espérant que ça marchera encore. Bien sûr que ça marchera, ça marchera et ça reproduira ce qu'on connait, le passé et sa mentalité. Abandonnez-la, elle ne mène nulle part ! Choisissez ce qui vous plaît et commencez à le faire. Faites-le pour vous ou pour d'autres, en faisant vous deviendrez de plus en plus compétent, vous construirez petit à petit votre réputation. Vous aurez une attitude agile envers vous-même.

Tous les mois, faites le plein de codes

Abonnez-vous à PROGRAMMEZ! le magazine du développeur

Nos offres fêtes

CPL Devolo dLAN 550

+

clé USB

+

+ Les anciens numéros disponibles
+ 1 numéro vintage de Programmez!

1 an → 64,90 € *

2 ans → 94,90 € *

Inclus 1 kit CPL Devolo dLAN 550, 1 clé USB contenant tous les n° de Programmez! depuis le n°100, les anciens numéros encore disponibles et 1 numéro vintage ! Quantité limitée. Cette offre peut s'arrêter à tout moment.

(*) Valeur du CPL 79,90 € - Valeur clé USB 29,90 € - Frais logistiques : 15,90 €

Nos classiques

1 an ..... **49 €**
11 numéros

2 ans ..... **79 €**
22 numéros

PDF ..... **30 €(*)**
1 an - 11 numéros
(*) Souscription sur le site internet

Etudiant ..... **39 €**
1 an - 11 numéros

Tarifs France métropolitaine

Vous souhaitez abonner vos équipes ? Demandez nos tarifs dédiés* : redaction@programmez.com (* à partir de 5 personnes)

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

ABONNEMENT à retourner avec votre règlement à :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex.

Offres fêtes : Programmez + kit CPL dLAN 550 + Clé USB+ anciens n° + 1 n° vintage

☐ ***Abonnement 1 an :*** 64,90 €
☐ ***Abonnement 2 ans :*** 94,90 €

☐ ***Abonnement 1 an au magazine :*** 49 €
☐ ***Abonnement 2 ans au magazine :*** 79 €
☐ ***Abonnement étudiant 1 an au magazine :*** 39 €
Photocopie de la carte d'étudiant à joindre

☐ M. ☐ Mme ☐ Mlle Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

email indispensable pour l'envoi d'informations relatives à votre abonnement

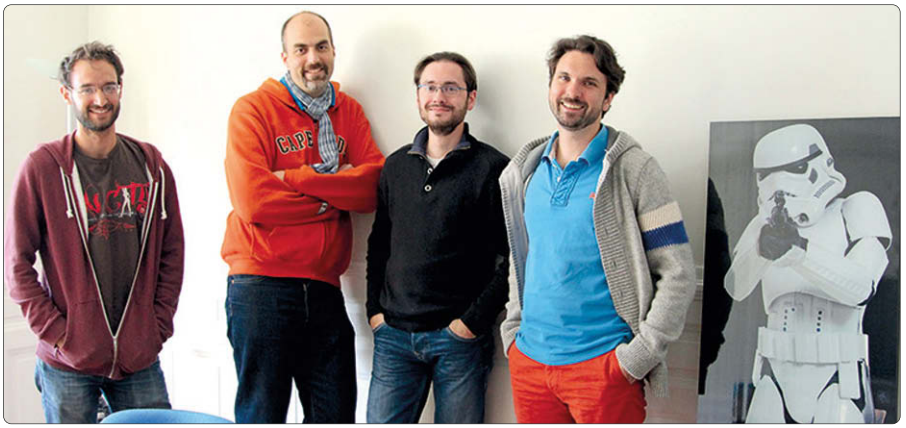
E-mail : @

☐ Je joins mon règlement par chèque à l'ordre de Programmez !
☐ Je souhaite régler à réception de facture
** Tarifs France métropolitaine*

PROG 191
***Offre limitée**, valable jusqu'au 31 décembre 2015*

De l'école à l'entreprise

Quand on fait des études, notre principale préoccupation est notre orientation. Il faut choisir une école, qu'elle soit publique ou privée, trouver un appartement si elle est loin de chez nous, vérifier que l'on peut y accéder avec nos précédents diplômes, et éventuellement passer les divers entretiens et/ou concours pour pouvoir y entrer. Et vous pouvez me croire si vous n'êtes pas encore passé par là : ce n'est pas une tâche facile ! Et le choix est encore plus ardu (tout du moins c'est l'impression que cela me donne) dans le domaine de l'informatique.



Quentin Metzler, Thomas Jaussoin, Nicolas Dominati et Olivier Berni et Gregory Lussiana (non présent sur la photo)

Kévin Sibué
Mail : ksibue@gmail.com
Site web : http://www.kevinsibue.com

Contrairement à beaucoup de métiers, le domaine de l'informatique reste assez nouveau dans l'éducation. Par conséquent les écoles ne sont pas nécessairement nombreuses notamment dans certaines régions. Qui plus est, il existe une multitude de métiers dans l'informatique, chacun disposant de son propre cursus. J'ai été personnellement confronté à ce problème, et ça très tôt, dès mon entrée en 1re. Comme vous le savez sûrement en fin de seconde, les élèves doivent choisir un BAC parmi de nombreuses possibilités : S, ES, L, STG, etc.

Mon plan au départ était de passer un BAC scientifique, dans le but de continuer sur un BTS informatique. Cependant il s'est avéré qu'une telle orientation pouvait être « dangereuse » si je souhaitais continuer dans ce domaine. Tout du moins suivant les études supérieures que je souhaitais effectuer. Alors après réflexion, j'ai choisi de faire une première STG, qui se trouvait être prioritaire pour le BTS que je souhaitais. Mon premier choix était fait, et sans le savoir, il allait définir une grande partie de mon avenir scolaire.

Mon second choix s'est effectué en terminale, où j'ai pu choisir le BTS SIO (Service Informatique aux Organisations), puisqu'à cause de mon précédent choix, les DUT ne m'ouvraient que difficilement leurs portes. Après avoir eu mon BTS haut la main, un troisième choix se présentait à moi : continuer mes études ou chercher du travail. J'ai choisi la première option, ce qui sous-entendait alors de trouver une nouvelle école pour faire ma troisième année (licence informatique).

Et cette fois-ci j'ai préféré m'orienter vers une école proposant l'alternance dans une entreprise pendant la formation. Je me suis tourné vers **Ecoris**, une école privée, implantée en Rhône-Alpes, proposant une multitude de formations, dont la licence *Concepteur de Système d'Information* (pour la première fois cette année). Mais qui dit alternance, dit entreprise. Je me suis donc mis à la recherche d'une société susceptible de me prendre en CDD durant la période de ma formation. Après quelques recherches, j'ai fini par trouver **Lunabee**, une startup d'une dizaine de personnes, éditrice de ses propres logiciels (*oneSafe* et Locky par exemple), proposant aussi le développement de logiciels sur mesure pour ses clients.

J'ai donc commencé cette nouvelle année scolaire à la fois à l'école, mais aussi en tant qu'employé chez **Lunabee**. Mais pourquoi ai-je choisi ce type de cursus ? Eh bien la réponse est assez simple : je voulais travailler, faire quelque chose de concret. Qui plus est, l'alternance possède un certain nombre d'avantages non négligeables à notre époque, comme notamment les années d'expérience. On nous demande souvent plusieurs années d'expérience pour intégrer une entreprise. Mais difficile quand on sort tout juste de l'école. L'alternance me permet de répondre à cela. Mais également d'acquérir un mode de travail que l'on ne peut apprendre qu'en entreprise. Et pour terminer, apprendre des technologies que je ne pourrais jamais voir à l'école, par exemple le développement pour iOS avec Objective-C ou Swift.

Je suis donc une sorte de « double formation » qui se complète. Chose qui aurait été difficile, pour ne pas dire impossible sans l'alternance. Mais ça me permet aussi de découvrir ce qu'est vraiment le métier de développeur dans une entreprise quand on est impliqué dans celle-ci. Et pour tous ceux et celles qui n'ont pas encore pu découvrir ce monde, je vous le dis, c'est très différent de ce que l'on nous apprend à l'école, et également très différent des idées préconçues que l'on peut avoir.

Mais il me reste encore et toujours la même question qui va se poser à moi : dois-je continuer mes études après ma licence ? Et si je continue quelles sont mes options ?

À vrai dire, la réponse reste assez confuse pour moi. Après tout, comme dit mon tuteur : « Peu importe ton niveau d'étude, en tant que développeur, que tu casses du C# ou du Java ou du Swift, ça ne compte pas ». Et je dois bien avouer qu'il a raison. Alors je dois maintenant me poser la question suivante : est-ce que je veux rester développeur ou monter en grade pour devenir chef de projet ou plus encore ? Eh bien pour ne rien vous cacher, je ne sais pas, tout du moins pas encore...

Les bénéfices de l'inner sourcing pour l'entreprise

Sophie Gautier,
spécialiste coordination et animations communautés chez **inno3**

Ce qui caractérise l'open source, ce ne sont pas seulement les licences ou l'ouverture et la mise à disposition du code, mais également le mode de fonctionnement de ses projets. Nous entendons par là, tant les processus de développement que la gouvernance, tout autant que les processus d'apprentissage et le partage. Si celui-ci n'est pas uniforme selon les communautés, il présente cependant une base qui peut être modélisée et adaptée en fonction de l'environnement dans lequel il est utilisé.

L'inner sourcing reprend les points clés des communautés open source, dans sa définition et sa méthodologie, afin de les appliquer en interne à l'entreprise ou l'organisation. À travers la culture, les pratiques et la méthodologie telles qu'utilisées et appliquées dans les projets open source, elle va donc s'appuyer sur la collaboration, la transparence, le mérite, l'autonomie. Ce dernier point étant un des points cruciaux qui rend l'application de l'inner sourcing parfois compliquée tant il demande une réforme complète de la culture et de la pensée du modèle de l'entreprise.

On va tout d'abord s'appuyer sur la collaboration entre les acteurs car elle permet de gagner du temps entre chaque version de développement. En bénéficiant du regard et de l'échange de plusieurs intervenants, cela permet d'optimiser donc de réduire les coûts de développement et d'améliorer la sécurité. Cette collaboration est rendue possible par la transparence des processus, ceux-ci étant accessibles à l'ensemble des participants. Toutes les décisions, les discussions, ainsi que les étapes de la vie du projet doivent être mises à disposition de tous. Cela offre à ceux qui prennent le projet en cours d'avoir accès au même niveau d'information que ceux qui sont déjà présents depuis le début. De même tous les outils doivent être accessibles. Cela permet de maintenir une culture et une co-création entre les équipes laissant place à un terrain fertile pour l'innovation au sein de l'entreprise. Cela favorise également un recrutement qualitatif en étant plus attractif et une fidélisation des talents en interne.

Ce partage permet une amélioration de la base de connaissances des individus, une mise en commun des méthodologies et une modélisation des processus à travers les équipes ainsi qu'un brain storming permanent appelant une plus grande créativité. Enfin l'autonomisation des acteurs permet une meilleure priorisation des travaux par les équipes, le développeur travaille sur ce qu'il préfère, cela encourage la méritocratie; par exemple, une reconnaissance par rapport aux contributions réelles, les tâches de gestion s'en trouvent donc allégées. Cette responsabilisation rend l'entreprise attractive pour les employés présents et futurs. Prendre des responsabilités, c'est s'investir, rechercher les meilleures opportunités, être créatif, performant et se sentir plus en prise avec la tâche à accomplir, la rendant ainsi plus intéressante.

Si l'on reprend les points clés des projets open source, la transparence est une des vertus premières. À travers des discussions ouvertes, des décisions accessibles, un code et une documentation ouverts, elle permet d'installer la confiance entre les participants, une meilleure gestion des ressources internes, leur répartition étant plus aisée et plus facile à monitorer, d'assurer une meilleure accessibilité et donc d'abaisser le ticket d'entrée du développement. En donnant aussi l'accès au client à la vie du projet, il peut ainsi évaluer sa qualité ainsi que celle du code.

Les méthodes de développement Agile ont revisité la façon dont les processus sont mis en place pour les remettre en perspective. Mais cela ne suffit pas, il faut aussi modifier la culture d'entreprise. Certains environnements de développement ne nécessitent plus l'écriture de spécifications détaillées par exemple, le plus souvent rendues inutiles une fois le code implémenté. Garder des traces au sein de releases notes, de blogs ou d'articles qui donnent tout de même accès à l'information et en même temps valorisent leur auteur. L'idée est d'abaisser au maximum les contraintes quand elles ne sont pas nécessaires, cela va du niveau de détails d'un rapport d'activité à la possibilité d'installer les outils nécessaires à son travail sans passer par une série d'autorisations.
La simplification de la chaîne décisionnelle augmente au maximum la réactivité des équipes. Les personnes les plus qualifiées lors de l'exécution d'une tâche sont en général rapidement identifiées au sein d'une équipe. Ce sont elles qui seront les mieux à même de décider pour l'équipe et de requérir un consensus au sein de celle-ci. Ce consensus est important, même si la justification est différente pour chacun, il faut que la décision soit collégiale. De même que cette simplification décisionnelle amène de la flexibilité, elle permet là encore de responsabiliser les acteurs, et par là, de les valoriser. Enfin, elle permet à tout un chacun de se perfectionner, d'aller plus loin dans ses connaissances en n'ayant recours au mentoring que de façon sporadique.

Ceci est rendu possible en maintenant en permanence les flux de communication. La communication est cruciale dans tous projets, c'est elle qui permet la coordination et garde la cohésion du groupe ainsi que la confiance entre les acteurs. Il n'y a rien qui doit venir par surprise et personne ne doit se sentir isolé dans son projet. Du fait du niveau de responsabilité endossé par chacun, la confrontation et la mise en commun des idées, décisions, etc. sont très importantes. Pour que cette communication puisse garder sa fluidité, il existe différents outils qui permettent de communiquer soit en différé soit en instantané. D'autres outils permettent le partage de l'information et sa mise à jour collaborative. Tous les processus, qu'ils soient de management de projet, d'industrialisation du produit, doivent être documentés, partagés et accessibles à tous, quelle que soit sa place.

La difficulté de l'inner sourcing réside dans le fait qu'elle entraîne un changement profond dans la représentation du monde de l'entreprise. L'un des points les plus importants notamment porte sur le comportement social et l'absence de hiérarchie. La notion d'égalitarisme est centrale et c'est sur elle que s'appuie l'ensemble des principes de production.
La considération que tous les êtres humains sont égaux dans leur statut social génère une décentralisation des pouvoirs. Les décisions sont alors basées sur un consensus et tout le monde part sur un pied d'égalité. Les retours d'expérience sont valorisés et les différents points de vue enrichissent la prise de décision. Enfin, cela permet de casser les barrières de 'classe' dans les contributions.

Je pratique le C++

Partie 5/5

Nous vous proposons une série d'articles sur la pratique de C++ pour que vous puissiez tous vous y mettre. Ce mois-ci on aborde la librairie Boost du site www.boost.org. Boost est une communauté de programmeurs C++ dont le but est de pousser les librairies qu'ils font vers la standardisation C++. Boost est une librairie C++ portable. Vous pouvez l'utiliser avec Visual C++ sur Windows et GCC ou CLang sur Linux.

Christophe PICHAUD | .NET Rangers by Sogeti
Consultant sur les technologies Microsoft
christophepichaud@hotmail.com | www.windowscpp.net

Comment obtenir Boost ?

Très simplement en allant dans la rubrique download du site et vous téléchargez un zip de 120 Mb ou un targz de 80 Mb. Vous décompressez votre archive et vous êtes presque prêt. La décompression vous donne une arborescence de 430 Mb. Il y a énormément de documentation HTML c'est la raison pour laquelle la taille du dossier est conséquente.

Compilation de Boost

Malgré le fait que Boost soit en majorité un ensemble de templates et de fonctions inline - donc juste les fichiers d'en-têtes suffisent - il y a des librairies qu'il faut compiler.

La première chose à faire est de lancer la compilation de la librairie. Cela se fait en deux étapes. Premièrement, il faut compiler le custom Make de Boost qui se nomme Bjam. Il faut lancer bootstrap.bat. Maintenant que Bjam est compilé, on va builder la librairie avec notre chaîne de compilation particulière à une plateforme. Je travaille avec Visual Studio 2013 donc voici ma ligne de commande :

```
bjam toolset=msvc-12.0 variant=debug,release threading=multi link=shared
```

Je fais une compilation en debug et en release et indiquant que je veux une librairie multi-thread et en mode dynamique (DLL). La compilation prend un peu de temps, mais disons qu'en 15 minutes c'est terminé.

Introduction à Boost.Serialization

Maintenant que nous avons listé l'ensemble des librairies disponibles dans Boost, nous allons passer au code ! Examinons une librairie très utile qui sait sérialiser et dé-sérialiser des classes en format binaire ou XML sans forcer. Cette librairie permet de sérialiser des containers, ce qui évitera de parcourir nos différentes collections pour sérialiser des éléments simples. Par contre, nous devons faire attention à une collision de nom dans l'espace de noms disponible. Je m'explique... Boost sérialise les boost ::string, les boost ::vector & co... Ce qui veut dire qu'il va falloir transvaser nos objets écrits avec la STL standard dans des objets Boost compatibles à moins que notre application soit écrite complètement avec Boost. C'est un choix d'architecture ! Nous allons partir sur une application que j'ai écrite pour ma fille en 2012. Il s'agit d'une application de dessin où l'on dispose des éléments prédéfinis sur une page comme des images, des rectangles, des ronds, des triangles, etc. Cette application a permis à ma fille de 8 ans à l'époque de savoir manier la souris et de jouer avec le Ribbon pour changer les caractéristiques des objets (couleur, épaisseur de traits, etc.).

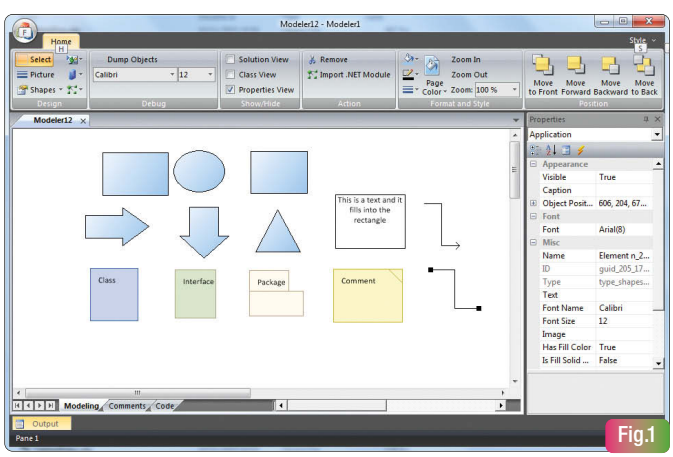
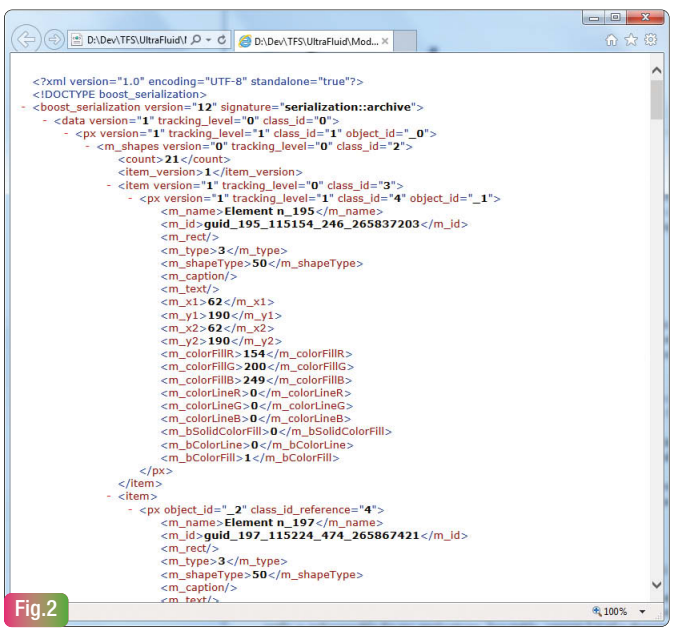


Fig.1

Le résultat de la sérialisation XML

Commençons par la fin... Le dessin est le suivant : Fig.1.

Le but du jeu est de sauvegarder ce dessin sous forme de document XML Fig.2. Ouvrons la documentation de Boost.Serialization pour prendre la chose du bon côté et pour arriver au résultat ci-dessus. Si vous avez l'œil, vous verrez que l'application que je vous propose est réalisée avec les Microsoft Foundation Classes (MFC) car elle me permet de faire un joli Ribbon et de mettre en place le support du Document/Vue qui me permet de faire une application qui charge/enregistre mes données. Bref, MFC rocks. Je crois d'ailleurs qu'on va refaire une série Je Pratique le C++ sous Windows en 5 épisodes pour que vous puissiez développer vous aussi des applications qui rockent ! On verra... revenons à nos moutons. Le XML. Boost.Serialization fonctionne avec ce que l'on appelle une Archive dans laquelle on a accès à des données. Il faut nourrir l'archive avec des données. J'ai donc une classe d'objets à dessiner et des objets. Le modèle de données est constitué de 2 classes : CSimpleShape et CShapeCollection.



```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
<!DOCTYPE boost_serialization>
<boost_serialization version="12" signature="serialization::archive">
  <data version="1" tracking_level="0" class_id="0">
    <px version="1" tracking_level="1" class_id="1" object_id="_0">
      <m_shapes version="0" tracking_level="0" class_id="2">
        <count>21</count>
        <item_version>1</item_version>
        <item version="1" tracking_level="0" class_id="3">
          <px version="1" tracking_level="1" class_id="4" object_id="_1">
            <m_name>Element n_195</m_name>
            <m_id>guid_195_115154_246_265837203</m_id>
            <m_rect/>
            <m_type>3</m_type>
            <m_shapeType>50</m_shapeType>
            <m_caption/>
            <m_text/>
            <m_x1>62</m_x1>
            <m_y1>190</m_y1>
            <m_x2>62</m_x2>
            <m_y2>190</m_y2>
            <m_colorFillR>154</m_colorFillR>
            <m_colorFillG>200</m_colorFillG>
            <m_colorFillB>249</m_colorFillB>
            <m_colorLineR>0</m_colorLineR>
            <m_colorLineG>0</m_colorLineG>
            <m_colorLineB>0</m_colorLineB>
            <m_bSolidColorFill>0</m_bSolidColorFill>
            <m_bColorLine>0</m_bColorLine>
            <m_bColorFill>1</m_bColorFill>
          </px>
        </item>
        <item>
          <px object_id="_2" class_id_reference="4">
            <m_name>Element n_197</m_name>
            <m_id>guid_197_115224_474_265867421</m_id>
            <m_rect/>
            <m_type>3</m_type>
            <m_shapeType>50</m_shapeType>
            <m_caption/>
            <m_text/>
```

Fig.2

La classe CSimpleShape

Cette classe contient les propriétés suivantes :

```
public:
   wstring m_name;
   wstring m_id;
   string m_rect;
   long m_type;
   long m_shapeType;
   wstring m_caption;
   wstring m_text;
   long m_x1;
   long m_y1;
   long m_x2;
   long m_y2;
   int m_colorFillR;
   int m_colorFillG;
   int m_colorFillB;
   int m_colorLineR;
   int m_colorLineG;
   int m_colorLineB;
   bool m_bSolidColorFill;
   bool m_bColorLine;
   bool m_bColorFill;
};

BOOST_CLASS_VERSION(CSimpleShape, 1)
```

La classe CShapeCollection

Cette classe contient la liste des elements :

```
public:
   vector<boost::shared_ptr<CSimpleShape> > m_shapes;
};

BOOST_CLASS_VERSION(CShapeCollection, 1)
```

C'est un vecteur de CSimpleShape tout simplement... Pour que ces éléments soient stockés sous forme de fichier texte, binaire ou XML il faut préciser quelques petites choses dans les classes. Il faut spécifier cela en private :

```
private:
   friend class boost::serialization::access;
   template<class Archive>
   void save(Archive & ar, const unsigned int version) const
   {
      ar & BOOST_SERIALIZATION_NVP(m_shapes);
   }

   template<class Archive>
   void load(Archive & ar, const unsigned int version)
   {
      ar & BOOST_SERIALIZATION_NVP(m_shapes);
   }

   BOOST_SERIALIZATION_SPLIT_MEMBER()
```

Cette partie de code nous montre le template à définir. Il s'agit d'une fonction template load et save qui travaille sur une Archive. La syntaxe pour enrôler une donnée dans l'archive se fait au travers de l'opérateur &. Cette partie de code enregistre le vecteur d'éléments. Mais pour que cela fonctionne, il faut aussi que le classe CSimpleShape enregistre ses éléments dans l'archive.

```
   template<class Archive>
   void save(Archive & ar, const unsigned int version) const
   {
      // ar & name;
      // ar & id;
      ar & BOOST_SERIALIZATION_NVP(m_name);
      ar & BOOST_SERIALIZATION_NVP(m_id);
      ar & BOOST_SERIALIZATION_NVP(m_rect);
      ar & BOOST_SERIALIZATION_NVP(m_type);
      ar & BOOST_SERIALIZATION_NVP(m_shapeType);
      ar & BOOST_SERIALIZATION_NVP(m_caption);
      ar & BOOST_SERIALIZATION_NVP(m_text);
      ar & BOOST_SERIALIZATION_NVP(m_x1);
      ar & BOOST_SERIALIZATION_NVP(m_y1);
      ar & BOOST_SERIALIZATION_NVP(m_x2);
      ar & BOOST_SERIALIZATION_NVP(m_y2);
      ar & BOOST_SERIALIZATION_NVP(m_colorFillR);
      ar & BOOST_SERIALIZATION_NVP(m_colorFillG);
      ar & BOOST_SERIALIZATION_NVP(m_colorFillB);
      ar & BOOST_SERIALIZATION_NVP(m_colorLineR);
      ar & BOOST_SERIALIZATION_NVP(m_colorLineG);
      ar & BOOST_SERIALIZATION_NVP(m_colorLineB);
      ar & BOOST_SERIALIZATION_NVP(m_bSolidColorFill);
      ar & BOOST_SERIALIZATION_NVP(m_bColorLine);
      ar & BOOST_SERIALIZATION_NVP(m_bColorFill);
   }
```

Le code du load et du save est le même. L'archive sait dans quel sens on travaille, soit en lecture, soit en écriture. Il faut noter que les MFC fournissent une infrastructure similaire avec les opérateurs << et >> ; je trouve que c'est plus lisible à mon goût, mais bon. Pour pouvoir compiler ce code, il faut disposer des en-têtes suivantes :

```
#define BOOST_SERIALIZATION_DYN_LINK TRUE
#define BOOST_ALL_DYN_LINK TRUE

#include <boost/smart_ptr/shared_ptr.hpp>
#include <boost/archive/tmpdir.hpp>
#include <boost/serialization/nvp.hpp>

#include <boost/archive/xml_iarchive.hpp>
#include <boost/archive/xml_oarchive.hpp>
#include <boost/archive/text_iarchive.hpp>
#include <boost/archive/text_oarchive.hpp>
#include <boost/serialization/shared_ptr.hpp>
#include <boost/serialization/base_object.hpp>
#include <boost/serialization/string.hpp>
#include <boost/serialization/list.hpp>
#include <boost/serialization/vector.hpp>
#include <boost/serialization/map.hpp>
#include <boost/serialization/utility.hpp>
#include <boost/serialization/assume_abstract.hpp>
using namespace boost;
```

L'écriture des données

Maintenant que nous avons nos classes qui savent utiliser une Archive, voyons le code qui donne l'ordre de faire Load ou Save. Vous allez voir, c'est très simple :

```
boost::shared_ptr<CShapeCollection> data(new CShapeCollection());
for( vector<std::shared_ptr<CElement>>::iterator i = m_objects.m_objects.begin() ; i!
=m_objects.m_objects.end() ; i++ )
{
    std::shared_ptr<CElement> pElement = *i;
    boost::shared_ptr<CSimpleShape> pNewElement(new CSimpleShape());
    pNewElement->m_name = pElement->m_name;
    pNewElement->m_id = pElement->m_objectId;
    pNewElement->m_type = pElement->m_type;
    pNewElement->m_shapeType = pElement->m_shapeType;
    pNewElement->m_caption = pElement->m_caption;
    pNewElement->m_text = pElement->m_text;

    CPoint p1 = pElement->m_rect.TopLeft();
    CPoint p2 = pElement->m_rect.BottomRight();
    pNewElement->m_x1 = p1.x;
    pNewElement->m_y1 = p1.y;
    pNewElement->m_x2 = p2.x;
    pNewElement->m_y2 = p2.y;

    pNewElement->m_colorFillR = GetRValue(pElement->m_colorFill);
    pNewElement->m_colorFillG = GetGValue(pElement->m_colorFill);
    pNewElement->m_colorFillB = GetBValue(pElement->m_colorFill);
    pNewElement->m_colorLineR = GetRValue(pElement->m_colorLine);
    pNewElement->m_colorLineG = GetGValue(pElement->m_colorLine);
    pNewElement->m_colorLineB = GetBValue(pElement->m_colorLine);

    pNewElement->m_bSolidColorFill = pElement->m_bSolidColorFill;
    pNewElement->m_bColorLine = pElement->m_bColorLine;
    pNewElement->m_bColorFill = pElement->m_bColorFill;

    data->m_shapes.push_back(pNewElement);
}

std::ofstream xofs(filename.c_str());
boost::archive::xml_oarchive xoa(xofs);
xoa << BOOST_SERIALIZATION_NVP(data);
```

La variable filename est remplie par l'ouverture d'une Common Dialog Windows SaveAs et toute la magie consiste à déclarer une xml_archive et à utiliser l'opérateur << pour sauver toutes nos données.

La lecture des données

La lecture des données est calquée sur l'écriture, on récupère les données et on les transvase dans la collection qui va s'afficher à l'écran :

```
boost::shared_ptr<CShapeCollection> data(new CShapeCollection());
// load an archive
std::ifstream xifs(filename.c_str());
assert(xifs.good());
boost::archive::xml_iarchive xia(xifs);
xia >> BOOST_SERIALIZATION_NVP(data);
// Clear existing shapes
m_objects.RemoveAll();
for( vector<boost::shared_ptr<CSimpleShape> >::iterator i = data->m_shapes.begin() ; i!
=data->m_shapes.end() ; i++ )
{
    boost::shared_ptr<CSimpleShape> pElement = *i;
    //AfxMessageBox(pElement->m_name + " " + pElement->m_id);

    std::shared_ptr<CElement> pNewElement = CFactory::CreateElementOfType((Element
Type)pElement->m_type,
                                                    (ShapeType)pElement->m_shapeType);
    pNewElement->m_name = pElement->m_name.c_str();
    pNewElement->m_objectId = pElement->m_id.c_str();
    pNewElement->m_caption = pElement->m_caption.c_str();
    pNewElement->m_text = pElement->m_text.c_str();
    pNewElement->m_pManager = this;
    pNewElement->m_pView = pView;

    CPoint p1;
    CPoint p2;
    p1.x = pElement->m_x1;
    p1.y = pElement->m_y1;
    p2.x = pElement->m_x2;
    p2.y = pElement->m_y2;
    pNewElement->m_rect = CRect(p1, p2);

    int colorFillR = pElement->m_colorFillR;
    int colorFillG = pElement->m_colorFillG;
    int colorFillB = pElement->m_colorFillB;
    pNewElement->m_colorFill = RGB(colorFillR, colorFillG, colorFillB);
    int colorLineR = pElement->m_colorLineR;
    int colorLineG = pElement->m_colorLineG;
    int colorLineB = pElement->m_colorLineB;
    pNewElement->m_colorLine = RGB(colorLineR, colorLineG, colorLineB);

    pNewElement->m_bSolidColorFill = pElement->m_bSolidColorFill;
    pNewElement->m_bColorLine = pElement->m_bColorLine;
    pNewElement->m_bColorFill = pElement->m_bColorFill;

    m_objects.AddTail(pNewElement);
    pView->LogDebug(_T("object created ->") + pNewElement->ToString());
}

// Redraw the view
Invalidate(pView);
```

Conclusion

L'utilisation de Boost et de Boost.Serialization est plutôt simple à appréhender. La documentation fournie est de bonne facture et les exemples sont légion dans la documentation. L'avantage de Boost.Serialization est que les archives sont portables... Donc pour faire du multiplateforme, c'est easy ! Le code complet de l'application de dessin est disponible ici : http://ultrafluid.codeplex.com .

Conclusion sur la série « Je pratique C++ »

Avec cet article, la série « Je pratique le C++ » s'arrête, mais je vais trouver un deal avec Programmez pour que nous ayons toujours du C++ dans Programmez. Tous les logiciels Microsoft sont faits en C/C++ à 90% et il ne serait pas normal de ne pas en parler. Le C++ n'est pas mort, loin de là. Avec C++11, le C++ a rajeuni et les standards C++14 et C++17 apportent leurs lots de nouveautés. Stay tuned comme dirait l'autre et à bientôt pour de nouveaux articles.

L'univers Xamarin

Si vous êtes au fait du développement mobile, le nom de Xamarin vous dit forcément quelque chose. Xamarin est une plateforme complète pour faire du développement multiplateforme, à partir d'un même code, écrit en C#. Il est tout naturellement très orienté mobile et a rapidement su devenir sans aucun doute l'environnement multiplateforme de référence.

Xamarin n'est pas un nouveau venu sur le marché. Il hérite en réalité d'un projet créé en 2001, le projet Mono. Mono était une alternative ouverte à la pile .Net / C# de Microsoft. Le leader du projet était Miguel de Icaza. Depuis Xamarin est né, de la volonté de deux hommes : Miguel de Icaza et Nat Friedman, de proposer des outils et des solutions packagées, adressant le marché mobile multiplateforme, aux entreprises et développeurs. La société compte environ 250 employés.
Comme vu plus haut, Xamarin permet de créer des applications iOS, Android, OS X à partir d'un seul code C#. La plateforme Windows étant adressée par Microsoft, elle est toutefois concernée par Xamarin si vos IHM sont développées en Xamarin.Form pour, ainsi, avec un seul code, adresser les trois plateformes. L'environnement transforme et gère les appels et les API selon la plateforme ciblée avec 100% de code en C#. 100% des API mobiles sont accessibles laissant ainsi aucune « infaisabilité ». Toutefois certaines limites sont posées par Xamarin.Form. Limites qui peuvent être levées comme le décrit un de nos articles plus loin. 100% de code écrit en C# ne signifie pas 100% de code réutilisable. Les spécificités de chacune des plateformes impactant naturellement ce niveau de réutilisation; l'éditeur annonce environ 75 % du code pouvant être partagé sur les différentes plateformes sans aucune limitation.
Ce niveau de réutilisation grimpe si vos IHM sont bâties sous Xamarin.Form acceptant ainsi certaines limitation. Xamarin génère des applications natives. A chaque fois, Xamarin développe et fournit une couche spécifique à chaque environnement.
Enfin, en dehors du Framework Xamarin, ce qui semble indispensable c'est de disposer d'une chaîne et d'une filière de développement complète. Les compléments tels que TestCloud ou les offres de backend mobile de Windows Azure, sont essentiellement des atouts pour vos développements mobiles. Dans ce dossier Xamarin, nous allons parler de Xamarin.Mac, Xamarin.Forms et de trucs et astuces à connaître.

Les différents outils

Xamarin a beaucoup évolué et propose désormais de nombreux outils. Les principaux sont :

Outils	Usages
Xamarin.iOS	Pour construire et générer des apps iOS. Supporte le SDK iOS, WatchKit. iOS 9 en cours de support. Nécessite XCode et iOS SDK.
Xamarin.Android	Pour générer des apps Android.
Xamarin.Mac	Pour les applications OS X
Xamarin Studio	IDE maison disponible sur OS X et Windows. Permet de créer et de gérer les projets de développement.
Xamarin for Visual Studio	Intégration avec Visual Studio (uniquement Windows)
Xamarin Test Cloud	Environnement de tests à la demande. Pour tester rapidement ses Apps mobiles. Supporte +1800 terminaux.
Xamarin Insights	Monitoring temps réel des applications
Xamarin.Forms	Outil pour construire des interfaces natives pour les différentes plateformes à partir d'un seul projet. Cette brique est gratuite pour Windows car nous n'avons pas besoin de Xamarin pour la filière Microsoft.

Xamarin Platform regroupe Xamarin.iOS / Xamarin.Android et Xamarin.Mac.

La question qui fâche : combien coûte Xamarin ?

Régulièrement, les développeurs reprochent à l'éditeur ses tarifs. Il faut avouer que pour un développeur indépendant, il y a intérêt à rentabiliser les développements. Actuellement, l'éditeur propose 3 souscriptions différentes : Indie (pour les indépendants), Business (essentiellement petits éditeurs et petites structures), Entreprise.

Les tarifs sont par développeur ET par plateforme.

	Indie	Business	Enterprise
Tarif	25 $ / mois	999 $ / année	1899 $ / année
Xamarin Studio	inclus	inclus	inclus
Déploiement Stores	oui	oui	oui
Xamarin.Forms	oui	oui	oui
Xamarin Test Cloud Access	oui	oui	oui
Intégration Visual Studio	non	oui	oui
Possibilité d'inclure Xamarin.Mac	non	oui	oui
Xamarin Université	+ 1995 $ / année	+ 1995 $ / année	+ 1995 $ / année

Coût pour 1 développeur (en entreprise) incluant iOS et Android avec la souscription business :

	Coût
Android (1 dév)	999 $
iOS (1 dév)	999 $
	Soit 1 998 $ par 1 an.

Vous pouvez commencer sur Xamarin avec la Starter Edition.

Et Test Cloud ?

Par défaut, les 3 souscriptions proposent un quota de tests sur Xamarin Test Cloud. Vous avez droit à 60 minutes, par mois et par développeur. Si vous avez besoin de tester plusieurs apps et d'un temps plus important, vous devrez choisir une souscription supplémentaire. La facture peut rapidement être lourde !

	Basic	Professional	Business	Enterprise
Tarif mensuel	1 000 $	5 000 $	8 000 $	12 000 $
Quotas	2 apps 200 heures support mail	4 apps 1000 heures support dédié	10 apps 1600 heures support dédié	20 apps 2400 heures support dédié
Xamarin Université	1 cours	1 cours	3 cours	5 cours

Démarrer avec Xamarin

La maîtrise du développement mobile sur différentes plateformes est plus ou moins complexe... Vouloir offrir les mêmes fonctionnalités aux utilisateurs n'est pas en soit le plus difficile, ce sont les technologies, les outils disponibles dans X ou Y plateforme et leurs SDK.

Andrés Talavera

Depuis l'arrivée de Xamarin, le développement d'applications natives avec un seul code devient envisageable ! Qui plus est, avec les technologies Microsoft.

Langage par défaut	iOS	Android	Windows Phone
Natif	Objective-C	Java	C#
Xamarin	C#	C#	C#

Les différents langages de développement avec/sans Xamarin

Prérequis

Xamarin permet donc de développer des applications mobiles sur iOS et Android depuis votre PC Windows ou de votre ordinateur Mac, avec Xamarin Studio ou Visual Studio.

Note : *spécificité pour le développement sur iOS, il est nécessaire de disposer d'un Mac avec les derniers outils de développement installés au préalable*

Xcode est disponible depuis l'App Store, le SDK et le simulateur iOS le sont dans le l'onglet des téléchargements, disponible dans les paramètres de l'IDE. Pour développer vos applications mobiles depuis Visual Studio, Xamarin propose des extensions : Xamarin.Platform for Visual Studio. Cette extension ne s'intègre pas dans les éditions Express.

Poste de développement	Plateforme Windows	Android	iOS
PC/Windows	SDK Windows Phone SDK Windows 8 SDK Windows 10	SDK Android JDK Android	Xamarin.iOS Xamarin Build Host Xamarin Studio
Mac/OS X		Émulateurs Android Xamarin Studio Xamarin.Android	Mac OS X > 10.9.3 Xcode SDK iOS Xamarin Studio Xamarin.iOS

Installations logicielles nécessaires au développement multiplateforme sur Windows et/ou OS X.

Concernant les émulateurs Android, il faudra les télécharger individuellement à partir d'Android SDK Manager. Les émulateurs fournis par Google sont très lents. Il existe des solutions alternatives, Genymotion, Android Player, etc.

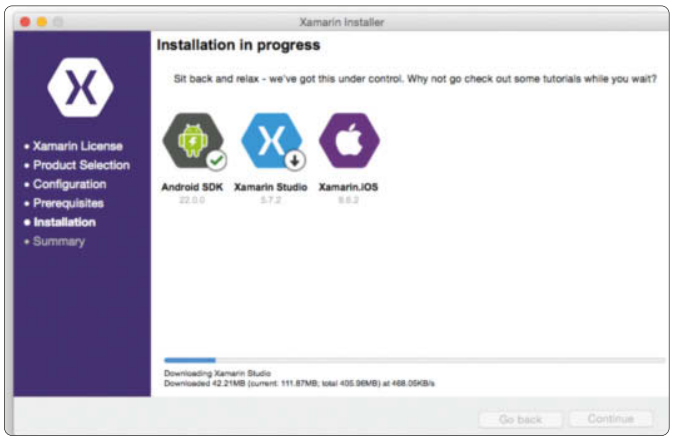
Information : *Les dernières versions de Visual Studio et de Xamarin fournissent des émulateurs nécessaires.*

Installation

Le téléchargement de Xamarin se fait depuis : http://xamarin.com/download. Un compte Xamarin est nécessaire pour valider une licence valide ou d'essai. L'installateur universel de Xamarin va vous demander de configurer l'emplacement des installations existantes, télécharger les composants nécessaires et/ou manquants, puis copier les fichiers dans votre ordinateur.

Démarrer un nouveau projet

Xamarin propose deux produits pour développer vos applications : Xamarin.Platform et Xamarin.Forms. La principale différence entre les deux est que Xamarin.Forms permet de partager le code de l'interface utilisateur, soit par code, soit de manière déclarative en XAML (juste le langage, ce ne



sont pas les mêmes contrôles que WPF, par exemple). Autre différence notable, Xamarin.Platform ne compile pas d'applications pour les plateformes Windows. À vous de faire en sorte de partager le code entre les plateformes mobiles.

Mon conseil

Dans le but de suivre les bonnes pratiques et de maintenabilité pour vos applications, il vaut mieux connaître les design patterns suivants : MVVM, inversion de contrôle (IoC), injection de dépendances...

Xamarin.Platform

Si vous souhaitez développer une application ne serait-ce que pour une plateforme, je vous recommande de créer, dans un premier temps, une PCL (Portable Class Library) qui contiendra le code métier. Puis, dans un deuxième temps, un projet par plateforme contenant le code UI et les bindings entre les vues et les ViewModels. MvvmCross ou MvvmLight sont deux bibliothèques téléchargeables dans le gestionnaire de packages NuGet et permettent une implémentation simple du pattern MVVM. Le code, commenté et documenté est disponible à l'adresse Internet suivante : http://bit.ly/gihubAT-XamPlatform-MVVM.

Xamarin.Forms

Si le partage du code de l'UI vous intéresse, créez un nouveau projet Xamarin.Forms et implémentez le pattern MVVM comme dans l'exemple présent mon Github : http://bit.ly/githubAT-XamForm-MVVM.

Archivage et travail en équipe

Si vous devez basculer entre PC et Mac pour développer vos applications mobiles, il existe un outil pris en charge par Xamarin Studio et Visual Studio : Git. Par habitude des produits et technologies Microsoft, j'utilise Visual Studio Online.

Publier vos applications

Depuis la version 7 de Xcode, Apple permet aux développeurs de déployer leurs applications de test sur leurs périphériques sans avoir besoin d'être enregistré dans un programme développeurs Apple. Plus d'informations à propos de l'enregistrement à l'adresse suivante : http://bit.ly/Xam-iOS-prov.

Développer des applications en C# sur Mac avec Xamarin.Mac

Lorsque l'on souhaite développer des applications sur OS X (Mac) et que l'on fait des recherches sur Internet, la plupart des réponses que l'on trouve indiquent qu'il est nécessaire d'utiliser Objective-C, ou plus récemment Swift, comme langage de programmation.

Stephane Cordonnier
MVP Office Development
http://blog.beecomedigital.com

Mais si l'on est un développeur .NET / C# et que l'on souhaite passer du côté obscur que représente le monde Apple pour nombre d'afficionados des technologies Microsoft, alors doit-on nécessairement abandonner son langage de programmation préféré ? Fort heureusement la réponse est non.
Très en vogue depuis un certain temps pour permettre de développer des applications mobiles multiplateformes (iOS, Android, Windows), Xamarin propose également de pouvoir faire du développement pour OS X grâce à sa solution Xamarin.Mac.

L'environnement de développement sur Mac

Comme sa déclinaison pour applications mobiles, Xamarin.Mac propose un environnement de développement sur Mac nommé Xamarin Studio, qui permet de pouvoir créer différents types d'applications allant de la simple application console (s'exécutant dans une fenêtre terminal) à une application native OS X capable d'utiliser l'ensemble des frameworks proposés par Apple (Foundation, AppKit, CoreData, SceneKit, etc.).
Pour créer une application, il n'y a rien de plus simple puisque Xamarin Studio propose un ensemble de modèles de projets permettant de répondre aux besoins les plus fréquents des développeurs Fig.1.
Une fois le projet créé et comme pour tout environnement de développement qui se respecte, on retrouve une vue de la solution permettant de retrouver l'intégralité des fichiers composant l'application Fig.2.

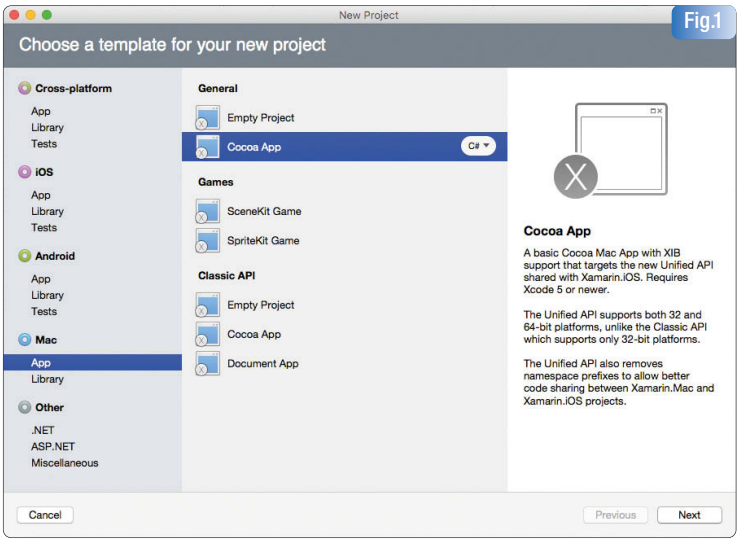


Fig.1

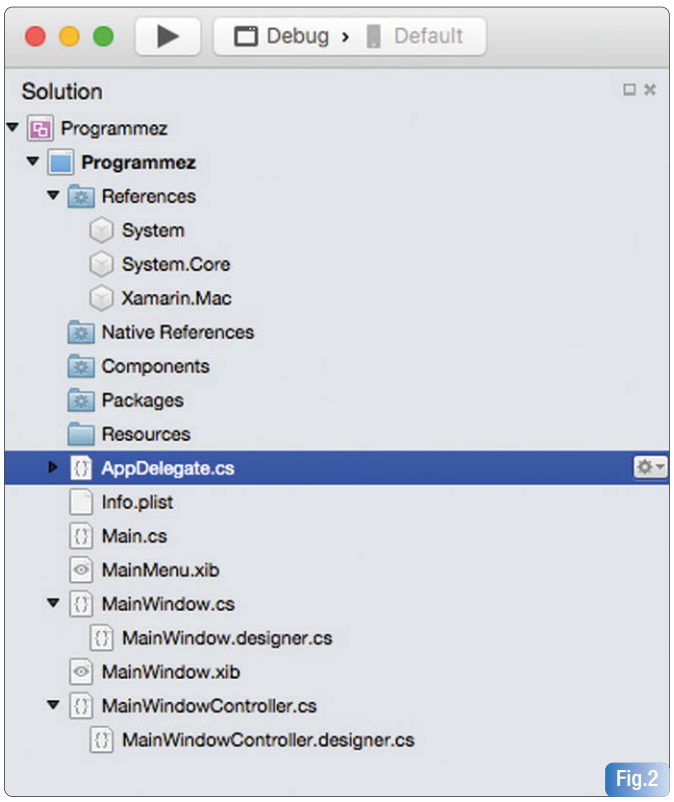


Fig.2

Si l'on s'attarde plus particulièrement sur le fichier *MainWindowController.cs* (le contrôleur principal de l'application basé sur MVC) créé par l'assistant de Xamarin Studio, on retrouve la structure classique d'une application .NET.

- Utilisation de MVC (Model View Controller) ;
- Fichiers XIB : Interface graphique (menus, fenêtres...)
- Fichiers CS : Logique application et métier ;
- Fichiers DESIGNER.CS : Générés automatiquement pour exposer au sein de .NET les composants (menus, boutons, listes...) .

```
using System;
using Foundation;
using AppKit;

namespace Programmez
{
  public partial class MainWindowController : NSWindowController
  {
    public MainWindowController(IntPtr handle) : base(handle)
    {
    }

    [Export("initWithCoder:")]
    public MainWindowController(NSCoder coder) : base(coder)
    {
    }
```

```
    public MainWindowController() : base("MainWindow")
    {
    }

    public override void AwakeFromNib()
    {
      base.AwakeFromNib();
    }

    public new MainWindow Window
    {
      get { return (MainWindow)base.Window; }
    }
  }
}
```

Les directives *using* nous permettent d'inclure les *namespaces* utilisés par l'application et l'on constate que les frameworks Apple (AppKit et Foundation) ont été encapsulés dans des wrappeurs .NET permettant de pouvoir manipuler toutes les APIs disponibles sur OS X de manière totalement transparente au sein du code C#.
C'est notamment le cas pour la classe *MainWindowController* qui hérite de *NSWindowController*, qui représente la classe de base de tous les contrôleurs de fenêtres utilisés dans les applications Cocoa.
On remarque également l'attribut *[Export("initWithCoder:")]* qui permet d'assurer la communication entre le monde .NET et le monde natif Apple (Objective-C). Cet attribut permet d'indiquer que le constructeur de la classe .NET est mappé sur la méthode *initWithCoder* qui existe en Objective-C.

Création d'une interface graphique

Comme évoqué précédemment, les fichiers XIB représentent les éléments d'interface graphique des applications Cocoa. Pour définir à quoi ressemblera l'application, il suffit de double cliquer sur un fichier pour basculer vers l'éditeur graphique qui permettra d'ajouter boutons, listes déroulantes et autres zones de saisies dans l'application.
Une subtilité existe toutefois si vous connaissez Xamarin pour iOS. Alors qu'il est possible d'éditer les interfaces graphiques directement depuis Xamarin Studio en utilisant des storyboard, l'édition des fichiers XIB d'une application Cocoa nous bascule automatiquement sous Xcode. Il est donc nécessaire d'avoir quelques connaissances sur le fonctionnement de celui-ci, notamment pour connecter les contrôles et les actions sur ceux-ci, à des éléments (IBOutlet / IBAction) qui pourront ensuite être manipulés au sein du code Fig.3.
Mais si l'on crée l'interface graphique sous Xcode, un environnement fait pour Objective-C et Swift, comment le lien se fait-il avec Xamarin Studio qui utilise .NET vous demanderez-vous ?
C'est là toute la magie de Xamarin car lorsque vous quittez Xcode pour revenir sous Xamarin Studio, ce dernier détecte les modifications que vous avez effectuées et synchronise les fichiers *.DESIGNER.CS en conséquence pour refléter vos modifications.
Ci-dessous, un exemple dans lequel un bouton a été ajouté à la fenêtre avec interception du clic sur ce bouton :

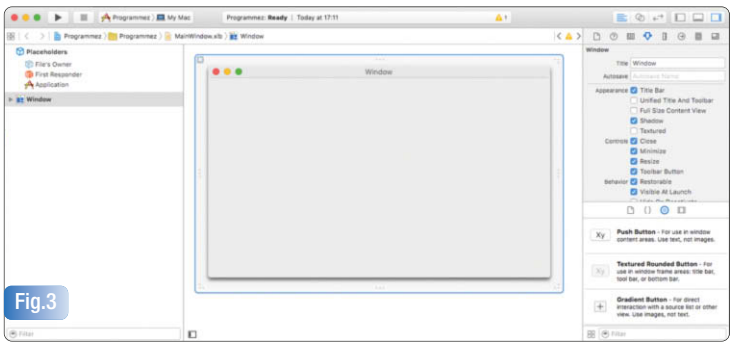


Fig.3

```
using Foundation;
using System.CodeDom.Compiler;

namespace Programmez
{
  [Register ("MainWindowController")]
  partial class MainWindowController
  {
    [Outlet]
    AppKit.NSButton MyButton { get; set; }

    [Action ("ButtonClicked:")]
    partial void ButtonClicked (Foundation.NSObject sender);

    void ReleaseDesignerOutlets ()
    {
      if (MyButton != null) {
        MyButton.Dispose ();
        MyButton = null;
      }
    }
  }
}
```

On voit que le bouton nommé *MyButton* est de type *NSButton*. L'attribut *[Outlet]* permet de faire le lien avec ce qui a été fait dans Xcode (connexion avec le contrôle qui a été nommé MyButton dans l'éditeur graphique).
La même chose a été faite concernant le clic sur le bouton avec la connexion de l'action nommée *ButtonClicked* déclarée dans Xcode, sur la méthode partielle *ButtonClicked* côté .NET.
Il ne reste désormais plus qu'à éditer le contrôleur pour implémenter la méthode partielle et effectuer le traitement nécessaire :

```
partial void ButtonClicked(NSObject sender)
{
    MyButton.Title = "Hello World";
}
```

Partager du code avec ses applications Windows et/ou mobiles ?

Ce qui fait tout l'intérêt d'une solution comme Xamarin, c'est notamment de pouvoir développer pour de multiples plateformes, aussi bien ordinateurs que mobiles, avec un seul et unique langage et de pouvoir réutiliser un maximum de code entre ces différentes applications.
Xamarin.Mac ne déroge pas à cette règle et l'on retrouve les mêmes mécanismes que sur Xamarin.iOS ou Xamarin.Android pour partager du code avec son application OS X. Il est ainsi possible de créer et partager du code via :

- L'utilisation de Class Library
- L'utilisation de Portable Class Library (PCL)
- L'utilisation de Shared Projects

Nous ne débattrons pas ici des avantages / inconvénients de chacune des solutions, de très nombreux articles existent sur ce sujet, mais toutes sont

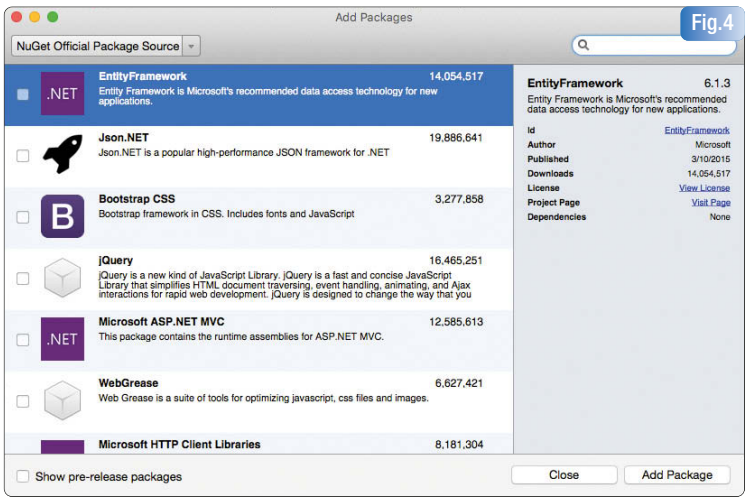


Fig.4

utilisables au sein de votre solution en fonction de vos besoins.

En revanche et en ce qui concerne les interfaces graphiques, pas de solution miracle, ce qui parait logique tant les différences d'ergonomie et d'usage sont différentes entre Windows, les mobiles et OS X.

Il est également bon de noter qu'il est possible d'utiliser au sein de son application OS X, les composants mis à disposition par des éditeurs tiers, vu le désormais très célèbre NuGET. Xamarin Studio propose en effet une intégration native de celui-ci dans son interface et vous pouvez ajouter des packages au sein de votre projet en quelques clics de souris Fig.4.

Bien évidemment, tous les packages disponibles sur .NET ne pourront pas être utilisés dans une application OS X (ex : un package destiné aux applications Silverlight) mais pour la majorité des références du marché (ex : Json.NET, HttpClient...) et qui simplifient la vie des développeurs, aucun souci particulier avec à la clé un énorme gain de temps dans les développements.

Aller plus loin

Bien d'autres possibilités sont également offertes par Xamarin.Mac pour aller au-delà du simple développement .NET sur OS X et la meilleure des manières de les découvrir est de se référer au portail des développeurs mis à disposition par l'éditeur :

http://developer.xamarin.com/guides/mac/getting_started/

Parmi les fonctionnalités qu'il est toutefois intéressant de citer, on peut commencer par la possibilité d'intégrer des librairies à l'origine écrite en Objective-C, au sein de son application .NET via la création de wrappeurs appelés *Binding Libraries*. Si vous devez réutiliser un composant métier sans avoir à le réécrire en totalité pour diverses raisons (ex : vous n'avez plus le code source d'origine), alors cette solution est faite pour vous.

On peut ensuite parler du vaste chantier engagé par Xamarin avec la création de ce qu'ils appellent les *Unified APIs* qui permettent de simplifier et d'assurer une meilleure compatibilité de partage de code entre iOS et OS X. Cette démarche s'est notamment inscrite au moment du passage au 64 bits obligatoire pour les applications iOS début 2015. Outre le fait d'uniformiser le nom des classes/méthodes entre les plateformes, il s'agit surtout de s'assurer que les types de données (32 u 64 bits) étaient manipulés de manière identique entre les plateformes.

Nous terminerons en évoquant la possibilité de pouvoir déployer ses applications sur le Mac App Store. En effet, Xamarin Studio contient tous les composants nécessaires ou capables de s'interfacer avec les outils d'Apple, afin de signer son application en vue de la déployer sur la boutique d'applications dédiées au Mac.

Développement avancé Xamarin : Les custom controls sous Xamarin.Forms

Xamarin.Forms est un accélérateur qui, au travers d'une librairie graphique, vous permet de construire des interfaces graphiques natives mobiles à partir d'un seul code de base. Elle complète l'école classique Xamarin qui vous permet d'accéder en C# à 100% des contrôles UI. Cette librairie vous propose actuellement plus de 40 composants : pages, contrôles... Des éditeurs tiers comme Telerik ou encore Infragistics, pour citer les principaux, proposent de compléter cette librairie.

Abdelkader Benabdi
Leader technique chez RedFabriQ depuis 2008. Expert Xamarin, Azure et plateformes .Net, Abdelkader a participé à de nombreux projets mobiles en passant par les technologies natives, cordova et Xamarin. Il promeut avec RedFabriQ cette technologie depuis 2011.

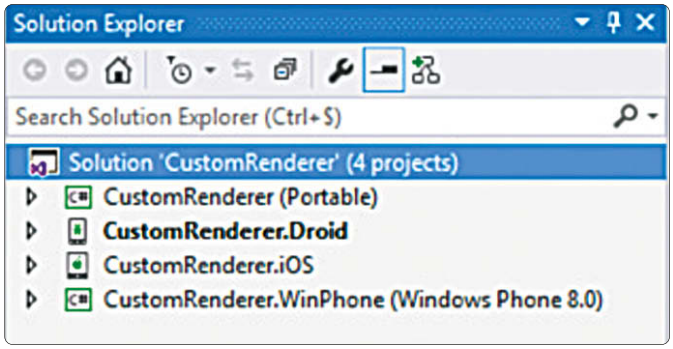
La philosophie initiale de XAMARIN, permettant d'accéder en C# à 100% de l'API du terminal cible, est respecté par Xamarin.Forms. Cet avantage vous permet sans encombre d'adopter les pratiques ergonomiques du ou des terminaux cibles. Il est par ailleurs également possible de débrayer en sortant de l'API de Xamarin.Forms pour adopter une stratégie de développement Hybrid, toujours en C#, en mixant du Xamarin.Forms et des appels aux contrôles UI natifs des terminaux mobiles ciblés. Concrètement pour développer sur un objet de type zone texte de saisie, si vous restez sous Xamarin.form vous allez manipuler l'objet « Entry ». Si vous débrayez et utilisez un contrôle natif alors en C# vous allez manipuler des UITextField sous iOS et EditText sous Android. Sachez que, lors de la phase de compilation, Xamarin.Form convertit l'objet « Entry » en UITextFild ou EditText.

Chez RedFabriQ nous utilisons la technologie Xamarin depuis 2011 (les années en informatique se comptent en année chien, dont 5*7 = 35 ans d'expériences ;-). Nous confirmons qu'il n'y a point de limitation pour développer des applications UI mobile complexe ou simple avec Xamarin. S'il y a une limitation c'est qu'elle est chez le développeur ou l'architecte !

Nous allons dans cet article expliquer comment débrayer et développer un custom control permettant au développeur de lever toutes les barrières.

Accès aux API natives

L'accès aux API natives en utilisant Xamarin.Forms se fait de trois façons : en utilisant les dependency service, les rendus graphiques personnalisés



ou à travers les plugins Xamarin. Les dependency services nous permettent de concevoir facilement des interfaces pour des implémentations spécifiques de sorte que vous pouvez accéder facilement à des fonctionnalités spécifiques de la plateforme comme l'accès à la caméra ou bien à l'API de géolocalisation dans votre projet commun ou portable.

Les interfaces graphiques créées avec Xamarin.Forms ont un rendu natif parce qu'elles sont natives. Elles sont affichées sur l'écran en utilisant le contrôle natif de chaque plateforme. Par exemple si vous utilisez un contrôle **Entry** dans Xamarin.Forms, sur iOS il sera affiché comme un contrôle **UITextField**, sur Android comme un contrôle **EditText** et sur Windows Phone comme un **TextBox**.

Les développeurs peuvent facilement changer ces rendus natifs en créant leurs rendus graphiques personnalisés qui peuvent consister à créer un simple contrôle ou bien à générer toute une page en natif.

Création d'un rendu graphique personnalisé

Malgré le fait que les pages et les contrôles fournis par Xamarin.Forms soient nombreux, il arrive un moment où on est obligé de définir nos propres rendus graphiques personnalisés. Regardons dans cet exemple comment on va définir un contrôle bouton avec une police de caractères personnalisée. Chaque rendu graphique personnalisé est défini par deux parties, une sous-classe de l'élément qu'on veut personnaliser défini dans le projet commun Xamarin.Forms de notre solution et une implémentation spécifique pour chaque plateforme pour définir l'apparence spécifique.

Structuration d'une solution Xamarin Forms

Comme vous pouvez le voir un projet Xamarin.Forms est structuré en général en 4 projets par défaut :

- **Projet portable** : projet commun entre toutes les plateformes ou on définit les pages et les contrôles Xamarin.Forms en plus des fonctionnalités métier, mais on peut aussi créer un autre projet portable pour séparer les fonctionnalités métier de la définition des écrans.
- **Projet Android** : projet client pour Android, il est le point d'entrée de notre application Android ou on peut implémenter tout ce qui est spécifique à la plateforme comme les rendus graphiques personnalisés.
- **Projet iOS** : projet client pour iOS, il représente l'entrée de notre application iOS ou on va créer tout ce qui est spécifique à la plateforme
- **Projet Windows Phone** : projet spécifique à la plateforme Windows Phone.

Création de la sous-classe du contrôle

Dans le projet partagé ou bien le projet PCL ou vous avez défini vos pages Xamarin.Forms, nous allons créer une sous-classe du contrôle qu'on veut personnaliser, comme suit:

```
public class ExtendedButton : Button
  {
    public static readonly BindableProperty CustomFontProperty =
      BindableProperty.Create<ExtendedButton, string>(
        p => p.CustomFont, 0);

    public string CustomFont
    {
      get
      {
        return (string)base.GetValue(ExtendedButton.CustomFontProperty);
      }
      set
      {
        base.SetValue(ExtendedButton.CustomFontProperty, value);
      }
    }
  }
```

Nous avons hérité de la classe Button de Xamarin.Forms pour redéfinir le rendu graphique du contrôle bouton qu'on veut personnaliser. On a défini aussi une propriété CustomFont qu'on pourra définir dans notre projet commun et réutiliser après dans la classe qui va définir le rendu graphique de notre contrôle bouton pour chaque plateforme.

Définir l'implémentation spécifique

Pour personnaliser l'apparence du contrôle, nous devons créer un rendu graphique sur chaque plateforme.

Chaque contrôle Xamarin.Forms a une classe de rendu graphique qu'on peut hériter comme la classe **ButtonRenderer**.

Ensuite, pour gérer les changements, on doit surcharger la méthode **OnElementChanged**. Toutes les modifications qu'on va apporter au contrôle bouton seront définis dans cette méthode dans la propriété Control, qui est juste une association à l'instance du contrôle natif.

Par exemple dans iOS ça correspond au contrôle **UIButton** et dans Android au contrôle **Button**.

L'implémentation du rendu graphique pour Android correspond à la classe suivante :

```
[assembly: Xamarin.Forms.ExportRenderer(typeof(ExtendedButton), typeof(Extended
ButtonRenderer))]
namespace CustomRenderer.Droid.Renderers
{
  public class ExtendedButtonRenderer : ButtonRenderer
  {
    protected override void OnElementChanged(ElementChangedEventArgs<Xamarin.
Forms.Button> e)
    {
      base.OnElementChanged(e);


      if(Control!=null)
      {
        ExtendedButton element = e.NewElement as ExtendedButton;

            if(!string.IsNullOrEmpty(element.CustomFont))
          Control.SetTypeface(Typeface.CreateFromAsset(this.Context.Assets,element.Custom
Font),
            Android.Graphics.TypefaceStyle.Normal);
      }
```

```
    }
  }
}
```

Nous avons hérité de la classe ButtonRenderer pour redéfinir notre rendu graphique pour le contrôle bouton dans Android.

```
public class ExtendedButtonRenderer : ButtonRenderer
```

Ensuite dans la méthode OnElementChanged nous avons récupéré la valeur de notre police de caractères depuis la propriété e.NewElement et l'avons affectée à l'instance Control qui correspond au contrôle bouton natif Android.

```
if(Control!=null)
    {
      ExtendedButton element = e.NewElement as ExtendedButton;

          if(!string.IsNullOrEmpty(element.CustomFont))
        Control.SetTypeface(Typeface.CreateFromAsset(this.Context.Assets,element.CustomFont),
          Android.Graphics.TypefaceStyle.Normal);
    }
```

Dans cet exemple on peut voir qu'on a accès à toutes les fonctionnalités natives d'Android.
Enfin, pour permettre à Xamarin.Forms de trouver correctement et d'utiliser le rendu graphique de notre contrôle personnalisé, nous devons ajouter l'attribut [assembly] au-dessus du namespace.
Le premier paramètre référence le contrôle Xamarin.Forms que vous souhaitez modifier, tandis que le second paramètre référence le rendu graphique spécifique à la plateforme pour le contrôle personnalisé, voici dans notre exemple à quoi ressemble cet attribut :

```
[assembly: Xamarin.Forms.ExportRenderer(typeof(ExtendedButton), typeof(ExtendedButtonRenderer))]
```

L'implémentation du rendu graphique pour iOS correspond à la classe suivante :

```
[assembly:Xamarin.Forms.ExportRenderer(typeof(CustomRenderer.Controls.ExtendedButton), typeof
(ExtendedButtonRenderer))]
namespace CustomRenderer.iOS.Renderers
{
  public class ExtendedButtonRenderer : ButtonRenderer
  {
    protected override void OnElementChanged(ElementChangedEventArgs<Button> e)
    {
      base.OnElementChanged(e);

      if (this.Control != null)
      {
        var element = e.NewElement as ExtendedButton;

        if (!string.IsNullOrEmpty(element.CustomFont))
        {
          this.Control.Font = UIFont.FromName(element.CustomFont, (nfloat)this.Element.FontSize);
        }
      }
    }
  }
}
```

Comme sur Android nous avons hérité de la classe ButtonRenderer pour redéfinir notre rendu graphique du bouton dans iOS comme suit :

```
public class ExtendedButtonRenderer : ButtonRenderer
```

Ensuite dans la méthode OnElementChanged nous avons récupéré la valeur de notre police de caractères depuis la propriété e.NewElement et l'avons affectée à l'instance Control qui correspond au contrôle bouton native iOS.

```
    if (this.Control != null)
    {
      var element = e.NewElement as ExtendedButton;

      if (!string.IsNullOrEmpty(element.CustomFont))
      {
        this.Control.Font = UIFont.FromName(element.CustomFont, (nfloat)this.Element.FontSize);
      }
    }
```

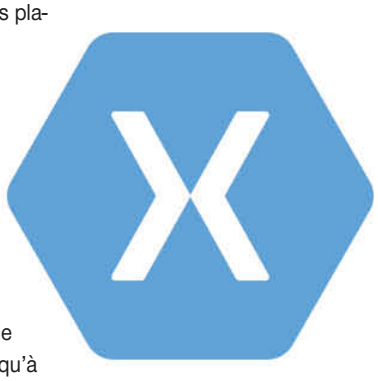
Comme vous pouvez le constater dans le code précédent on a accès à toute les fonctionnalités fournies par iOS.
Et de la même façon que sur Android, nous avons utilisé l'attribut [assembly] pour associer le nouveau rendu graphique au bouton utilisé dans Xamarin.Forms comme suit :

```
[assembly:Xamarin.Forms.ExportRenderer(typeof(CustomRenderer.Controls.ExtendedButton), typeof
(ExtendedButtonRenderer))]
```

Ce mécanisme d'implémentation des custom controls vous offre énormément de liberté pour apporter à vos applications l'expérience utilisateur qu'ils sont en droit d'attendre. Nous vous conseillons d'approfondir ces techniques car elles sont finalement loin d'être exotiques dans le développement d'applications Xamarin.

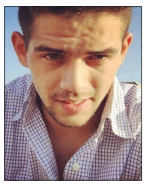
Conclusion

Xamarin.Forms est une librairie très puissante qui permet de créer des interfaces graphiques partagées entre les 3 plateformes mobiles Android, iOS et Windows Phone ; ceci est un avantage considérable de gain de productivité sur les trois plateformes mobiles phares.



En plus des contrôles et des pages fournis par Xamarin.Forms, la librairie reste ouverte à l'extensibilité sans limite en offrant un mécanisme de création de rendus graphiques personnalisés. Ce dernier, comme nous avons pu le voir, est facile et puissant. Et vous permet donc d'enrichir les contrôles Xamarin.Forms existants. On peut définir de simple personnalisations de contrôles jusqu'à créer des pages entière en natif. Ce qui nous permet de pouvoir créer des applications 100% natives avec le même code source. Autant au niveau de la couche logique métier qu'au niveau des écrans de nos applications.

Trucs et astuces



Andrés Talavera
développeur et formateur .NET, Web et Mobile

Performances

Les ressources de nos appareils mobiles sont parfois limitées. Entre autres, la batterie est souvent mise à l'épreuve. Les performances de notre application, et par conséquent l'expérience utilisateur, peuvent alors être dégradées. Les langages managés peuvent utiliser un gestionnaire de mémoire, un GarbageCollector (GAC). On peut alors optimiser l'utilisation de la mémoire. Mais attention, disposer et recréer des objets à chaque fois :

- Requiert plus d'allocations,
- Du temps au GarbageCollector,
- Peut provoquer des fragmentations de la mémoire,
- Peut être la source de fuites de mémoire.

Pour optimiser au maximum les performances de vos applications, il est donc impératif que vous connaissiez sur le bout des doigts le coût en mémoire des objets que vous créez et utilisez. Les outils de développement fournissent des *profilers.*

Information : *À l'exception de Xamarin.iOS avec les API Classic, tous les produits Xamarin utilisent un GarbageCollector de Mono, celui de SGEN.*

Limiter l'utilisation de ressources dans le temps

Lorsque notre programme doit allouer un espace mémoire à un nouvel objet, le GAC peut déterminer si un besoin de mémoire supplémentaire est nécessaire ou si une simple libération de mémoire suffit. L'utilisation du GargbageCollector nous évite la gestion d'allocation des objets, mais certains types d'objets nécessitent une attention particulière. Il peut être préférable de libérer explicitement les ressources sans attendre que le GarbageCollector intervienne, par exemple pour :

- Les connexions réseau,
- Les grands blocs mémoire
- Les fichiers

Le Framework .NET dispose de l'interface IDisposable qui permet de gérer la vie de nos objets et donc de les tuer dès lors que leur utilité devient nulle. Exemple d'utilisation d'un objet implémentant l'interface IDisposable :

```
using (var hw = new HelloWorld())
{
   hw.SayHello();
}
```

Réduire l'emprunte du GarbageCollector

Lorsque le GarbageCollector de SGEN commence, il arrête les threads de l'application pendant qu'il récupère la mémoire. L'interface peut donc être figée pendant quelques secondes. Cette pause dépend de deux facteurs :

- La fréquence de nettoyage du GarbageCollector
- La durée d'exécution de chaque nettoyage

Cela signifie que si de nombreux objets sont alloués et ne restent pas en vie, il y aura de nombreux nettoyages. Et vice-versa, lorsque les objets sont alloués et restent en vie, il y aura moins de nettoyages (mais plus longs).

Sources : http://www.mono-project.com/ http://developer.xamarin.com/guides

Architecture MVVM dans vos applications Xamarin

C'est devenu obsessionnel pour tout bon développeur : réutiliser le code plusieurs fois, dans différents contextes. Mais comment faire lorsque l'on code sur trois plateformes distinctes ?

Grâce à une PCL, on va partager notre code métier. Grâce au pattern MVVM, on va pouvoir, par exemple, gérer de manière simple les interactions à l'aide des ViewModels dans notre PCL ; et définir les vues dans chacune des plateformes. Il existe différentes librairies permettant l'implémentation simple du pattern MVVM, entre autres MvvmCross et MvvmLight. Ces deux dernières sont disponibles dans NuGet. La première est modulaire et complémentaire. Fonctionnant avec notamment le pattern d'injection de dépendances et d'inversion de contrôle, MvvmCross permet de créer d'autres librairies qui peuvent l'étendre.

Partagez votre code UI aussi !

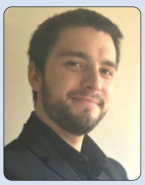
Source : http://msdn.microsoft.com/fr-fr/magazine/dn904669.aspx

Qui n'a jamais rêvé d'écrire une application mobile native pour plusieurs plateformes avec un seul code ? Xamarin apporte une réponse à cette problématique avec Xamarin.Forms. Avec Xamarin.Platform, pour partager un maximum son code, il était nécessaire de créer différents projets dont une PCL qui contient le code métier, partagé, et d'autres projets qui contiennent le code de l'interface utilisateur pour chaque plateforme. Xamarin.Forms permet de développer des applications mobiles natives en partageant le code de l'interface utilisateur (par code ou de manière déclarative avec le langage XAML). Il est possible de rajouter ses propres composants dans une plateforme dédiée avec des Custom Renderers par exemple (http://developer.xamarin.com/guides/ cross-platform/xamarin-forms/custom-renderer/renderers/).

RETOURS D'EXPÉRIENCE

Equipe de développement chez Technology and Strategy Strasbourg

Dès 2010, nos collaborateurs se sont montrés intéressés par le développement mobile. Les premiers projets iOS et Android ont été développés nativement. Nos clients réclamant souvent un développement simultané pour iOS et Android, nous avons souhaité proposer une solution pour mutualiser le code et gagner du temps. Nous recherchions une solution multiplateforme. En 2013 la branche "Web and Mobility" est née, renforçant notre positionnement sur les solutions mobiles. De par notre niveau Gold Partner avec Microsoft, nous avons tout naturellement choisi Xamarin. Nous étions et sommes toujours séduits par la flexibilité de la solution et sa communauté grandissante. Le développement en C# sous Visual Studio facilite les réalisations de nos équipes. Après plus d'une dizaine de projets déjà réalisés (applications Smartphone et tablette pour le B to B), nos clients sont aujourd'hui convaincus par les résultats obtenus. Le nombre de projets ne fait que progresser.



Antony Canut, développeur .NET chez SOAT

J'utilise Xamarin depuis un peu plus d'un an. Il me permet de réaliser des POC (Proof of Concept) très rapidement en cross-plateforme. Toujours grâce à ce logiciel, j'ai pu acquérir des compétences en Android et iOS, ce qui me permet de pouvoir aider ou de lire du code natif de ces plateformes très simplement. La solution Xamarin.Forms quant à elle, permet de ne pas me soucier des spécificités de chaque plateforme et ainsi de maintenir tout mon code très simplement.

Intégrer Azure Active Directory dans votre application Xamarin

Microsoft propose Azure Active Directory pour bénéficier d'un AD en Cloud, synchronisable ou pas, avec un AD classique. Par ailleurs, il met aussi à disposition la possibilité d'intégrer l'authentification AD dans une application Xamarin : exemple d'implémentation.

Thomas LEBRUN – *Consultant*
tlebrun@infinitesquare.com
http://blogs.infinitesquare.com/b/tom
http://blog.thomaslebrun.net
@thomas_lebrun

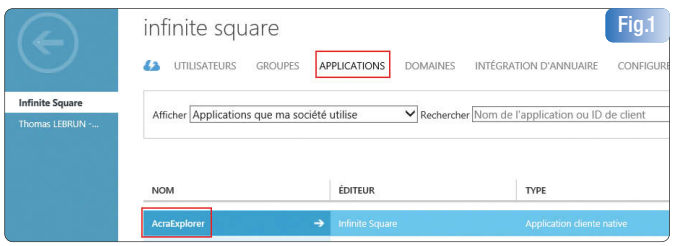
Azure Active Directory est la brique proposée par Microsoft dont le but est de permettre de disposer d'un annuaire Active Directory hébergé dans Azure. Et bien sûr, si vous disposez déjà d'un annuaire AD, vous pouvez le synchroniser avec la version « cloud ».

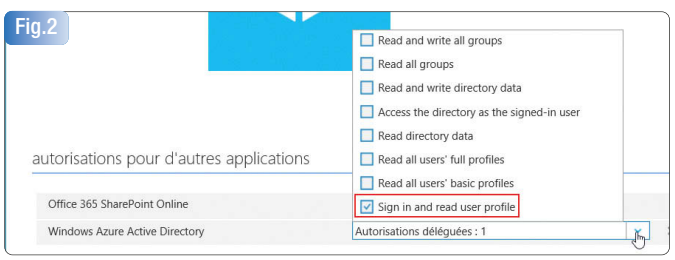
D'un autre côté, Microsoft met à disposition des développeurs le projet ADAL (Active Directory Authentication Library, accessible en téléchargement via Nuget ou ici : https://github.com/AzureAD/azure-activedirectory-library-for-dotnet), dont l'objectif est de permettre d'intégrer l'authentification Azure Active Directory dans une application, que ce soit une application .NET ou même une application Xamarin, puisqu'il en existe une version dédiée ! Si l'implémentation et l'utilisation d'ADAL au sein d'une application .NET ont déjà été couvertes sur Internet par différents articles de blogs ou autres, cela n'est pas le cas pour une application Xamarin, et c'est donc ce que nous allons voir par la suite.

La première étape dans la mise en place d'une authentification via Azure Active Directory consiste à créer une application, dans le portail Azure, qui sera rattachée à votre annuaire : Fig.1.

Dans les propriétés de cette application, vous devez à présent configurer les accès qui seront accordés aux utilisateurs lorsqu'ils seront authentifiés. Vous pouvez ainsi faire en sorte d'avoir accès aux profiles, aux informations des utilisateurs, etc. Si vous ne le faites pas, l'utilisateur recevra un message d'erreur lui indiquant qu'il n'est pas autorisé à accéder à la ressource demandée Fig.2. Une fois la configuration terminée, il est temps de passer au code ! Pour utiliser ADAL dans une application Xamarin (Xamarin.iOS ou Xamarin.Android), il faut d'abord rajouter le package Nuget nécessaire.

Il existe différentes versions des packages Nuget, certaines étant en état "Prerelease", indiquant qu'il s'agit d'une version Beta. Ces versions Beta intègrent, la majorité du temps, des corrections de bugs ou de nouvelles fonctionnalités.





Le package Nuget étant maintenant installé, il faut utiliser les APIs d'ADAL pour effectuer la connexion :

```
public async Task<AuthenticationResult> Authenticate(string clientId, string returnUri)
{
  string resource = "https://graph.windows.net"
  string authority = "https://login.windows.net/common"

  var authContext = new AuthenticationContext(authority);
  if (authContext.TokenCache.ReadItems().Any())
  {
    authContext = new AuthenticationContext(authContext.TokenCache.ReadItems().First().Authority);
  }

  var uri = new Uri(returnUri);
  var platformParams = new PlatformParameters((Activity)Forms.Context);
  var authResult = await authContext.AcquireTokenAsync(resource, clientId, uri, platformParams);

  return authResult;
}
```

Tout d'abord, on commence par récupérer un contexte d'authentification en passant en paramètre l'URL utilisée pour la connexion (dans notre cas, il s'agit de https://login.windows.net/common).

La partie « importante » concerne l'appel à la méthode **AcquireTokenAsync** : on passe en paramètre la ressource sur laquelle on souhaite se connecter pour en récupérer les informations (cela peut être le serveur de Microsoft pour récupérer les informations de profil, le serveur Sharepoint de l'entreprise, etc.). En retour de l'appel à cette méthode, on récupère une structure qui contient différentes informations :

- Un AccessToken ;
- Une date d'expiration ;
- L'identifiant du tenant Azure sur lequel l'utilisateur s'est authentifié au travers de votre code ;
- Une classe (UserInfo) qui contient les informations « de base » de l'utilisateur : le nom, l'identifiant, etc.

Sur ces différentes informations, c'est l'AccessToken qui est important, car c'est lui que vous allez devoir envoyer dans les headers de vos requêtes Web, lorsque vous interrogerez des ressources, pour faire en sorte de pouvoir y arriver de manière authentifiée. En effet, une fois connecté via la méthode **AcquireTokenAsync**, c'est en utilisant le jeton d'accès que vous pouvez accéder à vos ressources. En fonction des versions de la librairie ADAL que vous utilisez, vous pouvez rencontrer des différences/nouveautés. Ainsi, dans la version d'ADAL pour Xamarin.Android, le renouvellement du jeton d'accès (étape normalement obligatoire lorsque vous souhaitez utiliser le jeton, mais que celui-ci a expiré, d'où l'existence de la propriété « ExpiresOn ») est géré automatiquement par la librairie. Avec ADAL pour Xamarin, vous êtes donc en mesure de vous connecter, de manière extrêmement simple, à Microsoft Azure Active Directory afin, par la suite, d'accéder à des ressources qui sont protégées et utilisables uniquement lorsque l'utilisateur est authentifié !

Qu'est-ce que l'Open Data et comment l'utiliser ?

Le mouvement Open Data ou « données libres / données ouvertes » est une démarche visant à publier des données numériques en ligne accessibles et utilisables par tous. Cette approche consiste à publier sur des plateformes ouvertes des jeux de données détenus par les collectivités publiques, les organismes publics, les associations, les entreprises... Ces acteurs vont publier essentiellement des données intéressantes pour tous, par exemple des informations économiques, des rapports sur diverses pollutions, des cartes, des horaires, des agendas, etc.

Hugo Carnicelli

Richard Foucaud

Softfluent

Mais qu'on ne s'y méprenne pas, cette avancée est avant tout une obligation légale et donc encadrée strictement par la loi. En effet, un cadre juridique définit les informations qui peuvent être rendues publiques et celles qui ne le peuvent pas. Une donnée sera mise à la disposition du public que si elle répond aux critères suivants: Complète, Primaire, Opportune, Accessible, Exploitable, Non discriminatoire, Non propriétaire, Libre de droits, Permanente et Gratuite. En revanche, les données sensibles et à caractère personnel sont exclues de la démarche de l'Open Data.

L'objectif de l'open data est donc de constituer et d'entretenir une source d'informations de qualité et surtout fiable pour les chercheurs, les journalistes, les citoyens et nous, développeurs. Ces masses de données ouvertes vont pouvoir permettre l'émergence de nombreuses applications pratiques. L'enjeu pour nous sera donc de tirer profit de ces informations publiques et de mettre notre créativité au service de la société. Des besoins existent, il ne nous reste plus qu'à les trouver et transformer ces données brutes pour les rendre accessibles à tous et sur n'importe quel support.

Fig.1

CITYMAPPER - MAKING CITY USABLE

Open Data pour l'exemple

Pour illustrer tout ça, je vous propose de nous pencher sur l'exemple d'une application utilisant l'open data de manière déconcertante. Nous prenons le pari qu'après avoir lu ce petit paragraphe vous vous empresserez de l'installer sur vos smartphones Fig.1. CityMapper est une application disponible sur toutes les plateformes destinées aux calculs d'itinéraires géolocalisés dans les transports publics. L'équipe de développement anglaise a tout compris sur l'utilisation des données et les manie à la perfection. Au lieu de répartir les informations sous différentes applications, comme c'est souvent le cas pour les transports en île de France, CityMapper a réussi le pari de réunir les données d'une multitude d'applications (comme RATP, Vélib', Autolib', etc.) en une seule interface ludique et facile d'utilisation.

De ce fait, Citymapper va proposer le trajet le plus rapide, mais également de nombreux itinéraires alternatifs auxquels l'usager n'aurait pas forcément pensé (allant jusqu'à la marche à pied).

Par exemple l'application va vous conseiller de commencer votre trajet en métro puis de le terminer en Velib' sans oublier de vous indiquer, bien évidemment, le nombre de vélos disponibles lors de votre changement, et le nombre de places libres pour le déposer lors de votre arrivée à destination. Puissant non ? Ce produit nous prouve que la mutualisation et la bonne utilisation de l'open data peuvent donner naissance à des produits de grande qualité, prodiguant plus de services et une expérience utilisateur forte.

Quelques Statistiques sur l'Open Data

Chaque année, Global Open Data Index nous propose un classement complet par pays en fonction de sources de données libérées comme les transports, le budget et les dépenses de l'état, les résultats aux élections, les cartes nationales, les statistiques nationales, la législation du pays, les codes postaux ou encore les émissions polluantes. De par ces informations, Global Open Data Index nous fournit un tableau des pays modèles dans la libéralisation des données. En 2014 la France tient une place prédominante dans ce classement en se plaçant 3e (alors qu'en 2013 elle n'était que 12e). Cette ascension vient principalement de la libéralisation des codes postaux. Ainsi, on peut donc comprendre que la 1re place ne tient plus qu'à l'ouverture des données relatives aux dépenses du pays.

Où trouver des données ?

Pour commencer, en France il est possible de trouver bien des données sur des sujets variés sur le site http://data.gouv.fr. On remarquera que, par convention, la plupart des sites officiels permettant d'accéder à des données ouvertes sont généralement mis en ligne sur des sous-domaines nommés « data ». Il est donc relativement aisé d'en trouver un certain nombre grâce à une simple recherche Google : « inurl:data ». Cette recherche nous permettra de faire remonter des résultats tels que : https://www.data.gouv.fr/ - https://data.sncf.com/ - http://data.lesechos.fr/ http://data.worldbank.org/ - https://www.data.gov/ - https://data.oecd.org/ Fig.2.

Comment coupler Google Maps à une source de données ?

Nous allons dans cet exemple vous montrer comment lier une source de données, trouvée sur le site http://data.gouv.fr, à Google Maps uniquement en HTML et JavaScript. Première étape, commençons par télécharger le fichier suivant, grâce à la recherche « Liste des points de contact du réseau postal français » : https://www.data.gouv.fr/fr/datasets/liste-des-points-de-contact-du-

Fig.2

reseau-postal-francais. Fig.3. Nous allons ensuite nommer ce fichier « data.csv » et créer un fichier « index.html » qui va contenir le code source permettant d'afficher la carte de Google Maps. Attention, ces fichiers doivent être mis en ligne sur un serveur Web, même local, afin de pouvoir atteindre le fichier « data.csv » depuis le code JavaScript sans quoi notre appel AJAX sera refusé par le navigateur. Nous allons ajouter un élément qui va servir à contenir notre carte, en n'oubliant pas de lui donner une taille :

```
<div id="map-canvas" style="width: 500px; height: 400px;"></div>
```

Il nous faut ensuite ajouter comme référence la bibliothèque JavaScript suivante afin d'utiliser l'API Google :

```
<script src="https://maps.googleapis.com/maps/api/js"></script>
```

Pour finir, voici le code source permettant d'analyser le fichier « data.csv » et d'afficher les points de coordonnées qu'il contient sur la carte :

```
<script>

  function initialize() {
    var mapCanvas = document.getElementById('map-canvas');
    var mapOptions = {
      center: new google.maps.LatLng(48.86, 2.34),
      zoom: 12,
      mapTypeId: google.maps.MapTypeId.ROADMAP
    }
    var map = new google.maps.Map(mapCanvas, mapOptions);

    var request = new XMLHttpRequest();
    request.open('GET', '/data.csv', true);

    request.onload = function () {
      if (request.status >= 200 && request.status < 400) {
        var lines = request.responseText.split('\n');

        for (i = 1; i < lines.length; i++) {
          var columns = lines[i].split(';');
          // Point
          var marker = new google.maps.Marker({
            position: new google.maps.LatLng(columns[9], columns[10]),
            map: map,
            title: columns[2]
          });
        }
      } else {
        // error
      }
    };

    request.onerror = function () {
      // error
    };

    request.send();

  }

  google.maps.event.addDomListener(window, 'load', initialize);

</script>
```

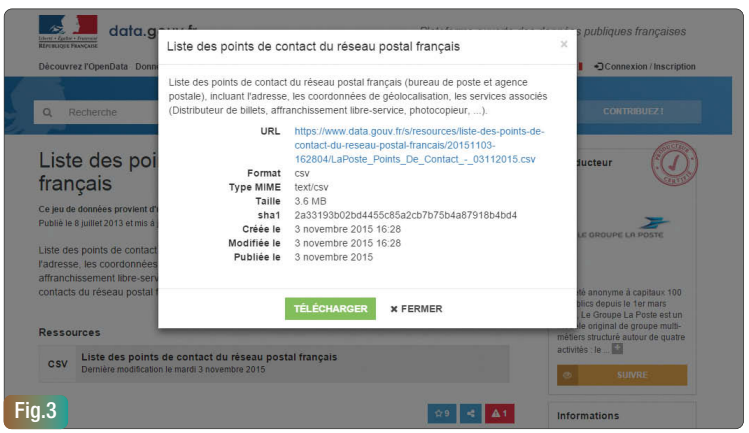


Fig.3

Fig.4

Ce code va simplement initialiser la carte puis lancer un appel asynchrone pour récupérer le contenu du fichier csv et enfin ajouter un marqueur pour chacune de ses lignes grâce aux colonnes de coordonnées. Et voici le résultat de notre développement pour le centre de la ville de Paris : Fig.4.

Pourquoi a-t-on besoin de données ?

Que ce soit dans l'univers des applications logicielles, mobiles ou encore Web, les données ont aujourd'hui une place prédominante dans la qualité du service que l'on propose à un utilisateur.

Admettons que l'on ait développé une application mobile de quizz sur les produits du terroir, il nous faut pour commencer une base de questions/réponses. Jusqu'à un certain point, il est bien entendu possible et généralement conseillé de construire cette base humainement. Mais que pouvons-nous imaginer de plus qu'une application de jeu, potentiellement multijoueur au travers d'un système de mise en relation automatique et de tableaux de scores ?

Et bien imaginons que nous souhaitions ajouter un scanner de code barre produit à notre application. Cela permettrait à un joueur de lancer un quizz sur le thème du produit qu'il scanne, ou bien tout simplement d'avoir des informations sur ce produit, autres que son prix et son nom qui sont très probablement inscrits sur l'étiquette. C'est là que les sources de données, fiables cela va sans dire, deviennent très importantes. Approfondissons notre sujet de code barre produit un moment... Fig.5.

Fig.5

Les 3 premiers chiffres du code servent à identifier le pays (300-379 pour la France, 400-440 pour l'Allemagne, etc.). Jusque-là pas de difficulté, cette codification ne variant pas souvent. Les 6 chiffres suivants servent à désigner le fabricant, et là déjà, cela se corse.

On trouvera facilement des applications Web permettant de retrouver cette information à la demande (http://www.gs1belu.org/ par exemple), mais rien d'automatisable via une API gratuite. Quant aux 5 chiffres suivants, il s'agit du code produit, le tout dernier chiffre étant un chiffre de contrôle. Là encore, le code produit n'est pas ou peu référencé de manière fiable dans les banques de données gratuites ou payantes. C'est en cela que les sources de données doivent progresser afin de permettre de proposer des services toujours plus complexes. Pour des raisons budgétaires évidentes, les sources de données payantes sont parfois en avance de phase sur certains secteurs.

Comment utiliser une source de données payante ?

Essayons maintenant d'utiliser l'API de la SNCF (http://data.scnf.com) en tant que développeur. Il vous faut tout d'abord vous inscrire sur le site https://data.sncf.com/api pour recevoir une clé d'accès par mail **Fig.6**.

Pour utiliser cette API, dont l'authentification se base sur le protocole HTTP, vous aurez besoin d'ajouter un header « Authorization » à votre requête. Voir cette documentation pour plus de précisions : http://tools.ietf.org/html/rfc2617#section-2.

Dans cette partie nous allons réaliser un exemple au travers d'une application Windows Universal. Notre programme devra :

- Récupérer la liste des gares et les afficher dans deux listes déroulantes (une pour le départ, une pour l'arrivée)
- Lancer via un bouton la demande de trajet
- Afficher un tableau (gares et horaires) pour afficher le résultat du trajet (le premier trajet de la liste si plusieurs possibles)

Pour commencer, nous allons créer notre projet Windows Store : **Fig.7**. Il nous faudra aussi installer le package NuGet de Newtonsoft pour la gestion du JSON. Maintenant, nous allons préparer notre gestion des appels à l'API. La classe « JourneyHandler » nous permettra de désérialiser la liste des gares :

```
using System.Collections.Generic;

namespace OpenDataSample
{
  public class StopPointsHandler
  {
    public List<StopPoint> places;
  }
}
```

La classe « StopPoint » quant à elle, représentant une gare, sera définie comme ceci :

```
namespace OpenDataSample
{
  public class StopPoint
  {
    public string id { get; set; }
    public string name { get; set; }
  }
}
```

Enfin, nous devons créer un ViewModel permettant de travailler en MVVM :

```
using System.Collections.Generic;
using System.ComponentModel;
```

Fig.6

```
namespace OpenDataSample
{
  public class ViewModel
  {
    public ViewModel()
    {
    }

    private ObservableCollection<StopPoint> _stopPoints;
    public ObservableCollection<StopPoint> StopPoints
    {
      get
      {
        if (_stopPoints == null)
        {
          _stopPoints = new ObservableCollection<StopPoint>();
        }
        return _stopPoints;
      }
    }
  }
}
```

Notre premier appel sera donc lancé au démarrage de l'application, plus précisément dans l'initialisation de l'écran principal « MainPage » (n'oubliez pas d'initialiser votre ViewModel par la même occasion) :

```
public MainPage()
{
  this.InitializeComponent();
  DataContext = new ViewModel();

  InitStopPoints();
}
```

La fonction « InitStopPoints » va lancer la requête HTTP et alimenter un objet de type « ObservableCollection<StopPoint> » (note : la variable « APIKey » contient la valeur de la clé reçue par mail suite à notre inscription sur le site de la SNCF) :

```
private async void InitStopPoints()
{
  using (var httpFilter = new HttpBaseProtocolFilter())
  {
    using (var httpClient = new HttpClient())
    {
```

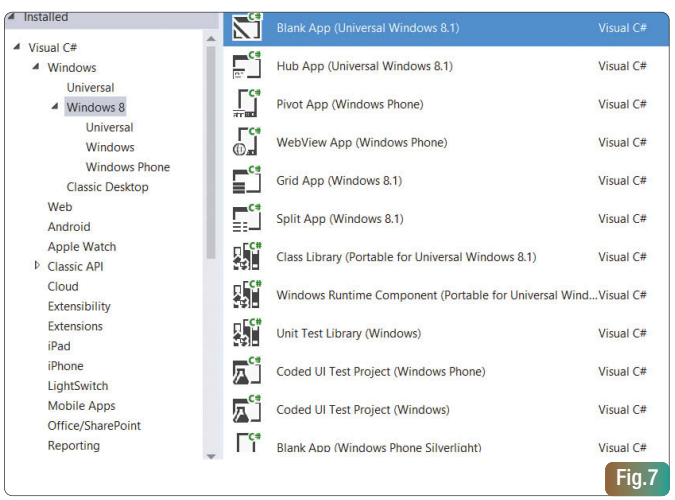


Fig.7

```
        // la clé d'API que nous avons reçu par mail encodé en base64
        byte[] byt = System.Text.Encoding.UTF8.GetBytes(string.Format("{0}:", APIKey));
        string password = Convert.ToBase64String(byt);

        // ajout du header
        httpClient.DefaultRequestHeaders.Add("Authorization", string.Format("Basic {0}", password));

        // endpoint de l'API
        HttpResponseMessage response = await httpClient.GetAsync("https://api.sncf.com/
v1/coverage/sncf/places?q=gare");
        if (response.StatusCode == HttpStatusCode.OK)
        {
          var responseAsString = await response.Content.ReadAsStringAsync();

          // récupération de la liste des gares
          var handler = JsonConvert.DeserializeObject<StopPointsHandler>(responseAsString);

          // mise à jour du modèle
          ((ViewModel)DataContext).StopPoints.Clear();
          foreach (var item in handler.places)
          {
            ((ViewModel)DataContext).StopPoints.Add(item);
          }

        }
      }
    }
```

Vous aurez besoin d'ajouter les Namespaces suivants :

```
using Newtonsoft.Json;
using System.Net;
using System.Net.Http;
using Windows.Web.Http.Filters;
```

Ne nous manque plus que la vue à implémenter pour afficher le résultat de l'appel, simplement deux labels de type « TextBlock » et deux listes déroulantes de type « ComboBox » :

```
<TextBlock x:Name="StartPointsLabel" HorizontalAlignment="Left" Margin="70,120,0,0"
TextWrapping="Wrap" Text="Gare de départ" VerticalAlignment="Top" FontSize="16"/>
<ComboBox x:Name="StartPoints" HorizontalAlignment="Left" Margin="70,153,0,0"
VerticalAlignment="Top" Width="520" ItemsSource="{Binding StopPoints}" DisplayMember
Path="name" />
<TextBlock x:Name="EndPointsLabel" HorizontalAlignment="Left" Margin="620,120,0,0"
TextWrapping="Wrap" Text="Gare d'arrivée" VerticalAlignment="Top" FontSize="16"/>
<ComboBox x:Name="EndPoints" HorizontalAlignment="Left" Margin="620,153,0,0"
VerticalAlignment="Top" Width="520" ItemsSource="{Binding StopPoints}" Display
MemberPath="name"/>
```

Fig.8

Fig.9

A ce stade, vous devriez pouvoir lancer votre application et afficher le résultat suivant : **Fig.8**. Et pouvoir sélectionner une gare dans chacune des listes déroulantes : **Fig.9**. Maintenant, nous allons pouvoir ajouter la partie la plus intéressante de l'application : la récupération du trajet. Pour commencer, nous allons ajouter un bouton permettant de lancer la recherche de trajet :

```
<Button x:Name="StartJourney" Content="Trajet" HorizontalAlignment="Left" Margin=
"1157,150,0,0" VerticalAlignment="Top" RenderTransformOrigin="0.646,0.582" Click=
"StartJourney_Click"/>
```

Ainsi que l'évènement « Click » correspondant :

```
private void StartJourney_Click(object sender, Windows.UI.Xaml.RoutedEventArgs e)
{
  GetJourney();
}
```

Avant de détailler la méthode « GetJourney », nous allons créer comme pour le premier appel les classes nous permettant de désérialiser le second appel. La classe « JourneyHandler » nous permettra de désérialiser le trajet proposé :

```
using System.Collections.Generic;

namespace OpenDataSample
{
  public class JourneyHandler
  {
    public List<Journey> journeys;
  }
}
```

La classe « Journey », représentant le trajet, sera définie comme cela :

```
using System.Collections.Generic;

namespace OpenDataSample
{
  public class Journey
  {
    public List<JourneySection> Sections;
  }
}
```

Et enfin, la classe « JourneySection », représentant une étape du trajet, sera définie ainsi (les deux dernières propriétés étant présentes uniquement dans un but d'affichage comme nous allons le voir par la suite) :

```
using System;

namespace OpenDataSample
{
  public class JourneySection
  {
    public StopPoint from { get; set; }
    public StopPoint to { get; set; }
    public string arrival_date_time { get; set; }
    public string departure_date_time { get; set; }

    public string Arrival
    {
      get
      {
        return DateTime.ParseExact(arrival_date_time, "yyyyMMddTHHmmss", System.
```

```
Globalization.CultureInfo.InvariantCulture).ToString("MM/dd/yy H:mm:ss");
        }
      }
      public string Departure
      {
        get
        {
          return DateTime.ParseExact(departure_date_time, "yyyyMMddTHHmmss", System.
Globalization.CultureInfo.InvariantCulture).ToString("MM/dd/yy H:mm:ss");
        }
      }
    }
}
```

Il nous faut aussi mettre à jour le ViewModel pour lui ajouter une propriété :

```
private ObservableCollection<JourneySection> _journeySections;
public ObservableCollection<JourneySection> JourneySections
{
  get
  {
    if (_journeySections == null)
    {
      _journeySections = new ObservableCollection<JourneySection>();
    }
    return _journeySections;
  }
}
```

Maintenant que notre représentation objet est en place, nous allons détailler la méthode « GetJourney » :

```
private async void GetJourney()
{
  using (var httpFilter = new HttpBaseProtocolFilter())
  {
    using (var httpClient = new HttpClient())
    {
      // la clé d'API que nous avons reçue par mail encodé en base64
      byte[] byt = System.Text.Encoding.UTF8.GetBytes(string.Format("{0}:", APIKey));
      string password = Convert.ToBase64String(byt);

      // ajout du header
      httpClient.DefaultRequestHeaders.Add("Authorization", string.Format("Basic {0}", password));

      // endpoint de l'API
      HttpResponseMessage response = await httpClient.GetAsync(
        string.Format(
          "https://api.sncf.com/v1/coverage/sncf/journeys?from={0}&to={1}&datetime={2}",
          ((StopPoint)StartPoints.SelectedValue).id,
          ((StopPoint)EndPoints.SelectedValue).id,
          DateTime.Now.ToString("yyyyMMddTHHmmss")
        )
      );

      if (response.StatusCode == HttpStatusCode.OK)
      {
        var responseAsString = await response.Content.ReadAsStringAsync();

        // récupération de la liste des gares
        var handler = JsonConvert.DeserializeObject<JourneyHandler>(responseAsString);

        // mise à jour du modèle
        ((ViewModel)DataContext).JourneySections.Clear();

        // par défaut, affichage du premier trajet possible
        if (handler.journeys.Count > 0)
        {
          foreach (var item in handler.journeys[0].Sections)
          {
            ((ViewModel)DataContext).JourneySections.Add(item);
          }
        }
      }
    }
  }
}
```

Vous noterez qu'à la différence du premier appel, outre le changement d'URL, nous transmettons maintenant 3 paramètres :

- L'id de la gare de départ
- L'id de la gare d'arrivée
- La date de la demande

Enfin, comme le premier appel, nous allons alimenter un objet cette fois de type « ObservableCollection<JourneySection> » récupéré depuis le premier trajet trouvé. Ne reste plus que la vue, nous allons donc utiliser une « ListView » pour afficher les différentes étapes du trajet : voir code complet sur le site de Programmez !. Vous noterez que le template est relativement simple, composé d'uniquement 4 « TextBlock » pour afficher respectivement les gares de départ et d'arrivée ainsi que leurs horaires respectifs. Et voilà ! Vous devriez maintenant pouvoir afficher le résultat d'un trajet : Fig.10.

Conclusion

Chaque jour, nous sommes confrontés à une multitude de données. Ces données font partie de notre quotidien, et sont aussi source de revenus pour l'économie. L'open data est encore récent et donc très peu connu par le grand public, mais deviendra un acteur incontournable du 21e siècle.

Cette démarche de libéralisation des données est une bonne chose à de nombreuses échelles. Tout d'abord, nos administrations vont pouvoir garantir la transparence de leurs actions publiques comme les projets, décisions, comptes publics et les rendre accessibles et compréhensibles par tous... sollicitant ainsi l'engagement citoyen. Nous pouvons espérer grâce à l'open data un renouveau des administrations qui vont pouvoir se moderniser et diminuer les délais de réactivité pour un futur meilleur. Enfin, nous verrons sans doute émerger de nouveaux services innovants et donc de nouvelles opportunités économiques, grâce à l'Open Data, qui est déjà considéré comme « l'or noir du 21e siècle ».

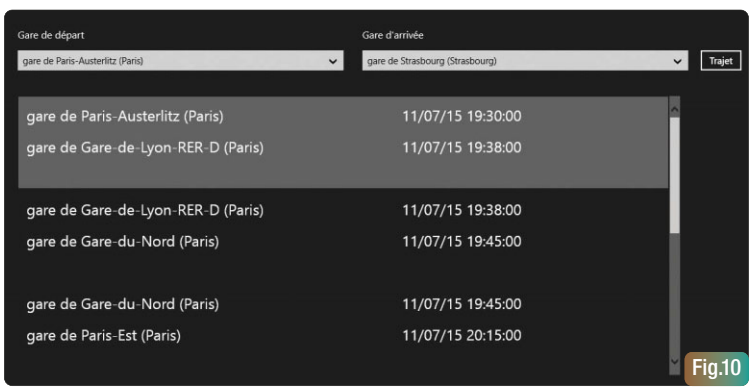


Fig.10

Comment consommer des données OpenData depuis une App Windows (Phone) 10

De nombreux organismes proposent depuis quelques années des données ouvertes. Les plus connus en France sont ceux du gouvernement sur http://data.gouv.fr ou https://data.sncf.com/. Pour retrouver une liste bien fournie d'API Open Data au niveau mondial je vous invite à consulter cette adresse : http://www.programmableweb.com/category/all/apis?keyword=opendata. Pour cet exemple, j'ai décidé d'être chauvin et de choisir des données de la ville de Bordeaux : http://opendata.bordeaux.fr/. Pour essayer de rester utile et ludique nous ne choisirons pas les budgets 2005 par nomenclature mais plutôt les emplacements des défibrillateurs.

Michaël FERY
Consultant Infinite Square
Blog : http://blogs.infinitesquare.com/b/mfery

PARCOURIR ET RÉCUPÉRER DES DONNÉES OUVERTES

Authentification et autorisations

Certaines API demandent aux services et applications de s'authentifier pour consommer leurs données. Cela permet de tracer la véritable utilisation de ces données mais également de limiter leur modification quand celle-ci est disponible. Dans l'exemple que nous avons choisi et pour plus de simplicité, aucune autorisation ne sera nécessaire. Les données sont disponibles directement depuis une url. Vous pouvez vérifier cela en tapant l'url du service directement dans votre navigateur :
http://odata.bordeaux.fr/v1/databordeaux/defibrillateurs/?format=json

Les formats

Comme vous pouvez le constater dans l'url nous passons un paramètre format avec la valeur Json. La plupart des API vont vous permettre de récupérer leurs données sous différents formats selon vos affinités et les usages que vous en aurez. Nous aurions ainsi pu saisir les url suivantes :
http://odata.bordeaux.fr/v1/databordeaux/defibrillateurs/?format=atom
http://odata.bordeaux.fr/v1/databordeaux/defibrillateurs/?format=kml
Encore une fois, pour plus de simplicité, nous utiliserons le format standard Json.

L'APPLICATION UWP

Comme le titre l'indique nous allons créer une application pour Windows 10, autrement appelée UWP pour Universal Windows Platform. Le langage utilisé sera le C#.
Pour cela, dans Visual Studio nous ferons donc « Fichier > Nouveau > Projet » et choisirons « Windows > Universal > Blank App » Fig.1.

Parcourir et récupérer des données ouvertes

Afin de pouvoir aisément manipuler les données Json récupérées nous allons utiliser une bibliothèque tierce appelée Newtonsoft.Json. Effectuez un clic-droit sur votre projet, puis sélectionnez « Manage Nuget packages ». Dans la fenêtre ouverte, choisissez d'installer le package Fig.2.
Ensuite nous allons devoir manipuler des données. Pour cela en général, nous créons nos classes afin de pouvoir désérialiser les données et pouvoir les manipuler. Afin de gagner du temps nous allons utiliser une fonctionnalité peu connue de Visual Studio.
Ouvrez la page correspondante à nos données dans votre navigateur :
http://odata.bordeaux.fr/v1/databordeaux/defibrillateurs/?format=json
Sélectionnez ensuite tout le contenu Json et effectuez un clic-droit – copier. Dans votre projet Visual Studio, créez une nouvelle classe, appelons-la Data. Dans ce fichier, supprimez la classe Data créée et effectuez un Edit > Paste Special > Paste JSON As Classes Fig.3.
Voilà du temps de gagné et des erreurs évitées. Bien sûr cette technique a des limitations et si votre Json est dynamique alors vous devrez mettre à jour vos classes vous-même.

Le code

Dans le code-behind de notre Page principale, c'est-à-dire dans le fichier MainPage.xaml.cs nous allons commencer par créer une constante correspondante à l'url du service OpenData :

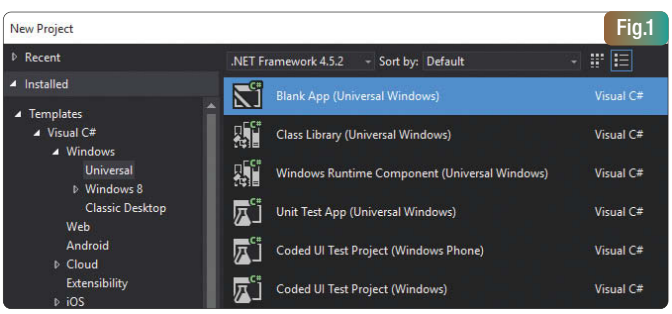


Fig.1

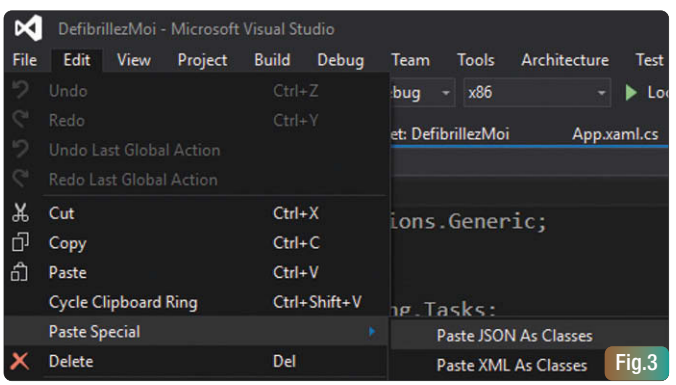


Fig.3

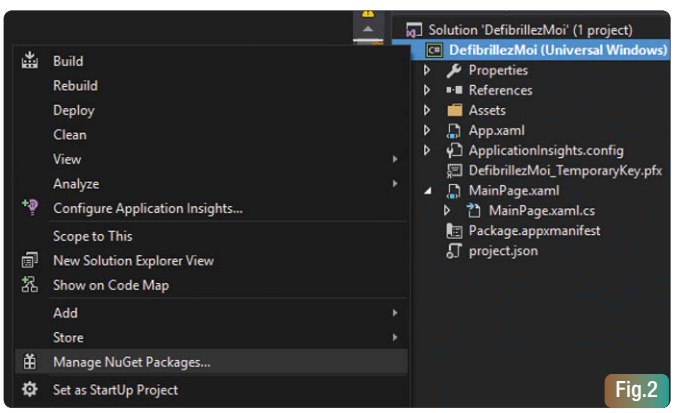


Fig.2

```
private const string DEFIBRILLATEURS_API_URL =
  "http://odata.bordeaux.fr/v1/databordeaux/defibrillateurs/?format=json";
```

Ensuite nous allons demander à notre page de charger dynamiquement les données des défibrillateurs en ajoutant un appel à une méthode asynchrone GetDefibrillateursAsync(); avant la fin du constructeur de la classe.
L'étape suivante est donc de créer cette méthode :

```
private async Task GetDefibrillateursAsync()
```

Dans cette méthode nous allons utiliser la classe HttpClient pour nous connecter et récupérer les données de la façon suivante :

```
var httpClient = new HttpClient();
var jsonResponse = await httpClient.GetStringAsync(DEFIBRILLATEURS_API_URL);
var result = await Task.Factory.StartNew(() => JsonConvert.DeserializeObject<Rootobject>
(jsonResponse));
```

Voilà nous avons nos données. Et maintenant ? On les affiche.

La représentation géographique

Dans le code xaml de notre page nous allons positionner un contrôle MapControl de la manière suivante :

```
<maps:MapControl x:Name="myMap" />
```

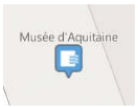
Malheureusement le designer ne permet pas un rendu direct de ce contrôle mais nous permet tout de même de le positionner et d'en estimer la taille à l'avance.
Si nous démarrons l'application maintenant nous nous apercevons que notre carte n'est pas du tout centrée sur la ville de Bordeaux comme cela semblerait logique, ni même zoomée correctement.
Pour effectuer le centrage et le zoom, nous allons modifier le constructeur de notre page afin d'y ajouter le code suivant :

```
myMap.Center =
  new Geopoint(new BasicGeoposition()
  {
    // Coordonées GPS de Bordeaux en degrés décimaux
    Latitude = 44.840,
    Longitude = -0.580
  });
myMap.ZoomLevel = 13;
```

Voilà qui est mieux, mais cette carte est bien vide. Nous allons donc devoir placer des éléments de type MapIcon pour chacun des défibrillateurs récupérés auparavant. Retournons dans le code-behind et améliorons notre méthode GetDefibrillateursAsync.
Pour cela nous allons boucler sur la liste des défibrillateurs et pour chacun, créer notre élément en lui affectant la latitude et la longitude correspondante. Nous ajouterons ensuite cet élément à la liste MapElements de notre MapControl :

```
foreach (var defib in result.d)
{
  var mapIcon = new MapIcon();
  var position = new BasicGeoposition
  {
    Longitude = double.Parse(defib.x_long),
    Latitude = double.Parse(defib.y_lat)
  };
  mapIcon.Location = new Geopoint(position);
  mapIcon.Title = defib.nom;
  myMap.MapElements.Add(mapIcon);
}
```

Voilà, il ne nous reste qu'à lancer notre application sur notre ordinateur pour en vérifier le bon fonctionnement.



Nous voyons ainsi apparaître des pictogrammes pour chacun des emplacements des défibrillateurs.
N'oublions tout de même pas que notre application est « universelle ». C'est-à-dire qu'elle pourra s'exécuter aussi bien sur un téléphone que sur un PC avec un écran de très grande taille.
Il nous faut penser responsive et pour cela le meilleur moyen serait d'utiliser les VisualStateManager.
Dans notre exemple nous allons nous contenter d'utiliser les ressources fournies par le Framework pour les tailles de titre et de laisser la carte prendre toute la place restante.

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

  <Grid x:Name="RootGrid" Margin="12,20,12,14">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <StackPanel Margin="0,0,0,10" Grid.Row="0">
      <TextBlock x:Name="description" Text="Defibrillez moi"
            Style="{StaticResource HeaderTextBlockStyle}"/>
      <TextBlock TextWrapping="Wrap"
            Style="{StaticResource SubtitleTextBlockStyle}">
        Emplacements des défibrillateurs de la ville de Bordeaux
      </TextBlock>
    </StackPanel>

    <maps:MapControl x:Name="myMap"
            Grid.Row="1"
            HorizontalAlignment="Stretch"
            VerticalAlignment="Stretch" />
  </Grid>
</Grid>
```

Fig.4.

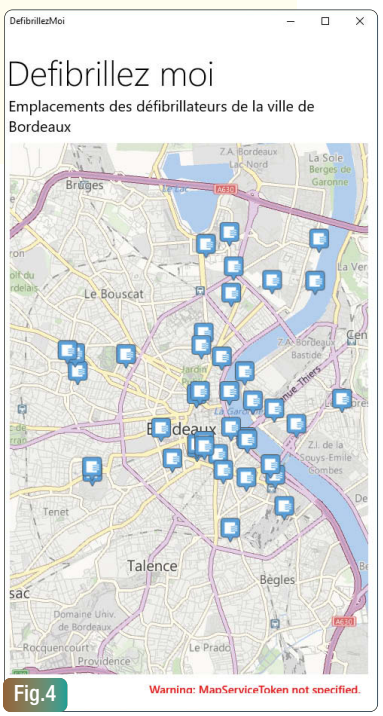


Fig.4

CONCLUSION

Nous avons vu ensemble qu'exploiter des données OpenData n'est pas très compliqué et que leur mise en application dans une application Windows 10 responsive n'est pas beaucoup plus sorcier.
Alors, à vous de lancer Visual Studio et de créer vos premières applications.

Visualisation cartographique de données ouvertes avec D3

Dans cet article, nous allons montrer comment exploiter différentes données ouvertes (ou open data) afin de les visualiser sur une carte en mode Web.

Thierry Templier
Restlet
ttemplier@restlet.com

Les sources de données

Le site data.gouv.fr fournit un important ensemble de données publiques utilisables librement. Il correspond à la principale source de données, même si certaines collectivités fournissent également des données de ce type. C'est le cas notamment de la région Ile-de-France via le site http://data.iledefrance.fr/.

Grégoire David met également à disposition librement un ensemble de ressources géographiques correspondant aux contours des entités administratives françaises (régions, départements, communes) via le projet « france-geojson » (https://github.com/gregoiredavid/france-geojson). Il ne faut pas non plus oublier OpenStreetMap (http://www.openstreetmap.org/). Ce dernier crée et fournit des données géographiques gratuites et libres du monde entier. Il vise à supprimer les restrictions techniques ou légales pour ce genre de données. Bien que l'outil permette de visualiser des cartes du monde en ligne, il fournit également au téléchargement les données géographiques utilisées par l'intermédiaire de la société Geofabrik. Il est possible de sélectionner une zone particulière. Pour notre article, nous utiliserons France > Ile-de-France.

Les différents formats

Il existe plusieurs formats cartographiques utilisés pour mettre à disposition des contours, des tracés ou des points :

- **ESRI Shapefile.** Format initialement développé par la société ESRI pour ses logiciels commerciaux et devenu un standard de facto largement utilisé, notamment par un grand nombre de logiciels libres. Il contient toute l'information liée à la géométrie des objets décrits.
- **GeoJSON.** Format ouvert d'encodage d'ensemble de données géospatiales simples basé sur JSON.

Pour une visualisation Web, nous rencontrons bien évidemment des problématiques de poids de données, afin que leur chargement ne soit pas trop long. À cet effet, un autre format, TopoJSON, est disponible. Il correspond à une extension de GeoJSON et encode les formes géométriques en segments de lignes afin d'éliminer les redondances et de fournir un contenu plus compact. Les fichiers de ce type sont 80% plus légers que leurs équivalents en GeoJSON.

Outillage

Plusieurs outils sont disponibles gratuitement sur Internet afin de manipuler ces formats et de les transformer. Le premier, GDAL — OGR, est le couteau suisse du géomaticien. Il supporte un grand nombre de formats tels que ESRI Shapefile et GeoJSON.

Le module typiquement utilisé de cet outil est « ogr2ogr » qui permet notamment de convertir un format ESRI Shapefile en GeoJSON et de tronquer des tracés.

La commande suivante décrit la manière de réaliser cette transformation :

```
templth@kerion ~ $   > ogr2ogr -f GeoJSON natural.json natural.shp
templth@kerion ~ $   >
```

Pour limiter des tracés, il est possible d'utiliser les paramètres commençant par « — clip ». Le paramètre « — clipsrc » permet notamment de limiter les formes à une zone décrite par quatre bords ou dans un fichier GeoJSON. Dans ce dernier cas, il peut s'agir d'un polygone complexe.

```
templth@kerion ~ $   > ogr2ogr -f geojson -clipsrc contours-departement.geojson cours-eau-rognes.g
eojson cours-eau.geojson
templth@kerion ~ $   >
```

L'autre outil que nous allons utiliser est « topojson ». Il permet de convertir des fichiers GeoJSON au format TopoJSON.

```
templth@kerion ~ $   > topojson -p -o cours-eau-rognes.topojson cours-eau-rognes.geojson
bounds: 2.39237962397136 48.1205460104373 44 3.473201728453402 49.11403222400534 (spherical)
pre-quantization: 0.120m (0.00000108°) 0.110m (9.94e-7°)
topology: 2783 arcs, 47212 points
post-quantization: 12.0m (0.000108°) 11.0m (0.0000994°)
prune: retained 2783 / 2783 arcs (100%)
templth@kerion ~ $   >
```

Maintenant que nous avons tout ce qu'il nous faut aussi bien au niveau de l'outillage que des données, nous pouvons attaquer l'implémentation de la visualisation sur carte en mode Web.

CAS D'UTILISATION

Dans cet article, nous allons afficher sur une carte avec différents niveaux de couleurs, la population de chaque commune du département de Seine-et-Marne. Cela permettra de rapidement visualiser les zones les plus habitées du département.

Sources de données utilisées

Pour construire notre carte, nous agrégerons deux types de données :

- Les données cartographiques permettant de définir les bases de notre carte comme les contours de départements et de communes ainsi que les tracés des cours d'eau;
- Les données structurées fournissant le nombre d'habitants par commune ainsi que la localisation des principales communes du département Fig.1.

La corrélation entre ces deux types de données se fera soit par l'identifiant INSEE des communes ou par leurs localisations (longitude / latitude). Les données géographiques contiennent en plus des contours des propriétés

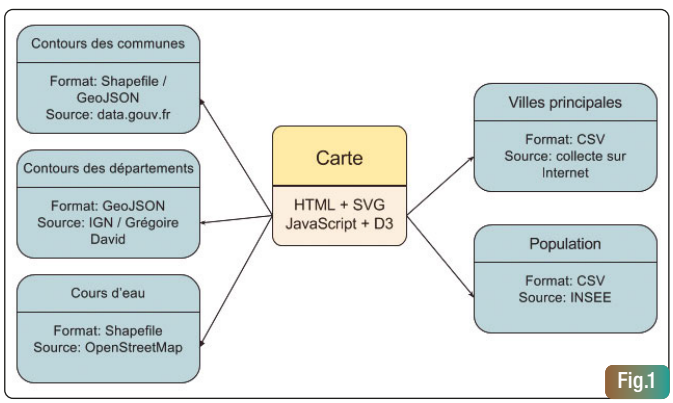


Fig.1

décrivant chaque forme. Dans le cas des contours des communes, les propriétés que nous allons exploiter possèdent la structure suivante :

Identifiant	Description
C_CAINSEE	Le code INSEE de la commune
L_CAB	Le nom de la commune
B_BOIS	S'il s'agit d'un bois ou d'une forêt
L_BOIS	Le nom du bois ou de la forêt

Concernant les données relatives au nombre d'habitants, nous allons tirer parti des colonnes suivantes :

Identifiant	Description
Code département	Le code du département
Code commune	Le code de la commune au sein du département
Population totale	Le nombre d'habitants pour une commune

Comme vous pouvez le voir, le code INSEE n'est pas présent explicitement dans le second jeu de données, mais peut être déduit en concaténant les champs « Code département » et « Code commune ». Une autre petite adaptation à réaliser est la suppression des espaces dans le champ « Population totale » afin de transformer ses valeurs en entier.

Préparation des données

Comme vous avez pu le constater, les données ouvertes récupérées sur Internet sont brutes. Elles sont de plus beaucoup plus larges que ce dont nous avons besoin et ne sont pas forcément dans un format utilisable directement dans une application Web. Il y a donc une phase préalable de préparation des données pour les rendre exploitables dans notre application, ainsi que décrites dans la figure suivante Fig.2.

MISE EN ŒUVRE DE NOTRE CARTE AVEC D3

D3 est un outil utilisable dans une page Web afin de visualiser des données sous forme de graphiques ou de cartes. Bien qu'il reste assez bas niveau, il offre néanmoins des facilités afin de les construire en peu de lignes de code. Il permet également d'utiliser des données TopoJSON afin de les convertir en des structures affichables par la technologie sous-jacente utilisée (SVG ou canvas HTML 5).

Préliminaires

La première chose à faire est d'inclure les bibliothèques D3 et TopoJSON dans notre page HTML qui va contenir la carte. Cela peut être fait simplement en se basant sur le site d3.org, comme décrit ci-dessous :

```
<html>
  <head>
  </head>
  <body>
    <script src="//d3js.org/d3.v3.min.js"></script>
    <script src="//d3js.org/queue.v1.min.js"></script>
    <script src="//d3js.org/topojson.v1.min.js"></script>
  </body>
</html>
```

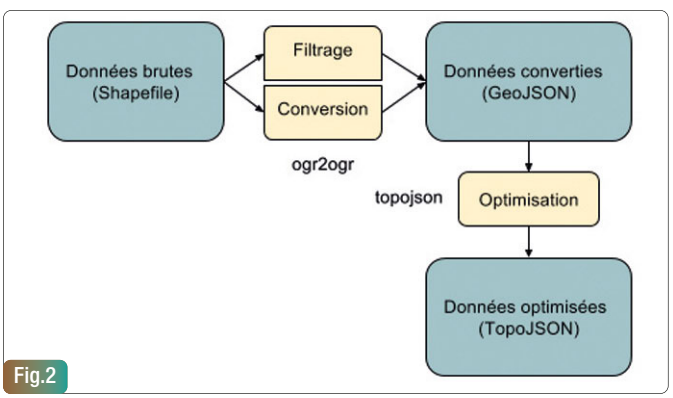


Fig.2

Nous afficherons les contenus TopoJSON en nous basant sur le support de SVG des navigateurs et de D3. À cet effet, nous devons maintenant créer un bloc SVG.

```
var largeur = 960, hauteur = 900;

var svg = d3.select('body')
    .append('svg')
    .attr('width', largeur)
    .attr('height', hauteur)
    .attr('preserveAspectRatio', "xMidYMid meet")
    .attr('viewBox', '0 0 960 900');

var carte = svg.append("g").attr('id', 'carte');
var departements = carte.append("g").attr('id', 'departements');
var communes = carte.append("g").attr('id', 'communes');
var formes = carte.append("g").attr('id', 'formes');
var communesLabels = carte.append("g").attr('id', 'communes-labels');
```

Comme vous pouvez le voir, la carte a été structurée en sous-ensembles par l'intermédiaire d'éléments SVG « g ».

Dans le contexte des cartes, il convient de spécifier une projection en fonction de l'affichage souhaité. Pour la France, la projection conique conforme de Lambert, ou plus simplement, la projection de Lambert est communément utilisée. Elle se configure simplement avec la fonction « d3.geo.conicConformal".

```
var projection = d3.geo.conicConformal()
    .center([2.754071, 48.8379229])
    .scale(50000)
    .translate([width / 2, height / 2]);
var path =  d3.geo.path().projection(projection);
```

La projection permet également de définir le centre de la carte ainsi que son échelle. La variable "path" configurée avec la projection permet de construire les constituants de la carte dans le SVG. Elle sera utilisée par la suite.

Affichage des contours des communes

Maintenant que nous avons les fondations de notre carte, affichons les contours des communes de Seine-et-Marne. Nous sommes partis d'un fichier contenant les contours de toutes les communes de la région parisienne. Nous devons donc restreindre celles incluses dans le département. Les outils "ogr2ogr" et "topojson" vont nous servir ici.

```
Terminal
templth@kerion ~ $   > ogr2ogr -f geojson -clipsrc contours-departement.geojson contours-communes.
geojson contours-region.geojson
templth@kerion ~ $   > topojson -p -o contours-communes.topojson contours-communes.geojson
bounds: 2.392372 48.120542 3.556545 49.117551907960646 (spherical)
pre-quantization: 0.129m (0.00000116°) 0.111m (9.97e-7°)
topology: 1960 arcs, 55636 points
post-quantization: 12.9m (0.000116°) 11.1m (0.0000997°)
prune: retained 1960 / 1960 arcs (100%)
templth@kerion ~ $   >
```

Ne pas oublier le paramètre "– p" pour l'inclusion des propriétés dans le fichier TopoJSON.

Nous avons maintenant uniquement les données souhaitées. Nous pouvons les afficher avec D3. Une de ses forces est la possibilité d'ajouter des éléments et d'appliquer des propriétés sur un ensemble d'éléments avec du chaînage d'appels de méthodes. Il fournit à cet effet un mécanisme puissant de sélection d'éléments avec les méthodes "selectAll", "data" et "enter". Dans le code suivant, nous ajoutons un élément "path" pour chaque donnée de notre fichier TopoJSON.

```
d3.json('donnees/contours/contours-communes-77.topojson', function(err, donneesContours) {
  var contoursCommunes = topojson.feature(donneesContours,
            donneesContours.objects['contours-communes-77']);

  communes.selectAll('path').data(contoursCommunes.features)
          .enter()
          .append('path')
          .attr('d', path)
          .style('fill', 'none')
          .style('stroke', 'black')
          .style('stroke-width', '.3px');
});
```

C'est un bon début, mais nous pouvons faire encore mieux ! Ajoutons un contour en gras tout autour du département. Pour ce genre de cas, TopoJSON fournit la fonction "topojson.mesh". Cette dernière permet d'identifier les contours aux extrémités. Elle prend à cet effet en paramètre une fonction. Si nous sommes dans ce cas, ses deux paramètres sont égaux. Nous

pouvons alors le sélectionner en retournant “true”. Pour ces éléments, nous pouvons donc surcharger les propriétés des bordures pour les faire plus épaisses.

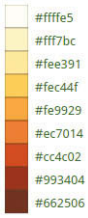
```
d3.json('donnees/contours/contours-communes-77.topojson', function(err, contours) {
  // ...

  communes.append("path")
          .datum(topojson.mesh(contours,
            contours.objects['contours-communes-77'], function(a, b) {
              return a === b;
          }))
          .attr('d', path)
          .style('fill', 'none')
          .style('stroke', 'black')
          .style('stroke-width', '3px');
});
```

Ajoutons également un ombrage ainsi que les contours des départements d’Ile-de-France d’une manière similaire, mais sans les communes. Nous ne surchargeons pas l’article avec ce code, mais vous pourrez le retrouver dans le code source joint. Voici le résultat ! Fig.3.
Dans le context de SVG, les styles suivants sont utilisés :

Style	Description
fill	La couleur de fond d’une zone
stroke	La couleur du contour d’une zone
stroke-width	L’épaisseur du contour d’une zone
opacity	Le niveau de transparence

Appliquons maintenant sur cette carte les informations relatives à la population.



Visualisation de la population par commune

La première chose à faire est de choisir la palette de couleurs que nous souhaitons utiliser. Color Brewer (http://colorbrewer2.org/) propose de telles palettes pour une utilisation dans des cartes. Nous choisissons ici la palette “YlOrBr” avec des nuances orangées.

```
var color = d3.scale.threshold()
  .domain([ 50, 500, 1000, 2500, 5000, 10000, 15000, 20000 ])
  .range(['#ffffe5', '#fff7bc', '#fee391', '#fec44f', '#fe9929',
                     '#ec7014', '#cc4c02', '#993404', '#662506']);
```

Nous déduisons ensuite les intervalles correspondants directement à partir des données. Plus le nombre est élevé, plus la couleur sera foncée.
Les informations relatives à la population des communes sont stockées dans un fichier CSV qu’il est nécessaire de charger par l’intermédiaire d’une requête AJAX à l’instar de ce qui a été fait pour les contours des communes. Comme nous avons désormais plusieurs requêtes, il convient d’attendre le chargement de toutes les données avant de les traiter. D3 fournit un support à cet effet par l’intermédiaire du module “queue” (https://github.com/mbostock/queue). Quand toutes les données ont été reçues, une fonction est appelée avec celles-ci en paramètres.

```
queue()
  .defer(d3.json, 'donnees/contours/contours-communes-77.topojson')
  .defer(d3.csv, 'donnees/population/dep77-2012.csv')
  .await(ready);

function ready(error, contours, population) {
  if (error) throw error;

  // ...
}
```

Nous allons maintenant construire un objet JavaScript associant le code INSEE des communes avec la population correspondante. De cette manière, la détermination de la couleur pour la zone sur la carte sera plus simple.

```
var populationParId = {};

population.forEach(function(d) {
  populationById[d['Code département']+d['Code commune']] =
          parseInt(d['Population totale'].replace(/\s/g, ''));
});
```

Nous n’avions défini précédemment aucune couleur dans l’attribut “fill” (valeur “none”) des éléments “path” pour les contours des communes.
Une des forces de D3 est de permettre d’utiliser aussi bien une valeur qu’une fonction calculant dynamiquement cette valeur en fonction de l’élément courant. Cette dernière approche correspond typiquement à notre besoin puisque la couleur va désormais dépendre de la population de la commune.

```
communes.selectAll('path')
        .data(contoursCommunes.features)
        .enter()
        .append('path')
        .attr('d', path)
        .style('fill', function(d) {
          if (d.properties.B_BOIS === 'Oui') {
            return '#01DF3A';
          } else {
            return color(populationById[d.properties.C_CAINSEE]);
          }
        })
        .style('stroke', 'black')
        .style('stroke-width', '.3px');
```

Voyons maintenant le résultat. La couleur de fond d’une commune est désormais fonction de sa population Fig.4.

Ajout de titre et de légende

Afin de rendre notre carte plus lisible, nous pouvons ajouter un titre ainsi qu’une légende pour les couleurs utilisées. Il s’agit simplement d’ajouter en SVG avec D3 des rectangles et des zones de texte. Le code suivant décrit la construction de la zone de titre.

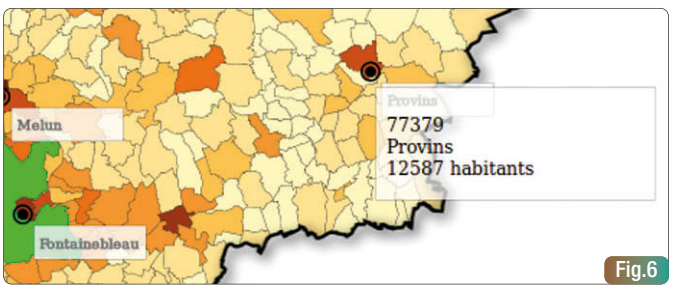


Fig.6

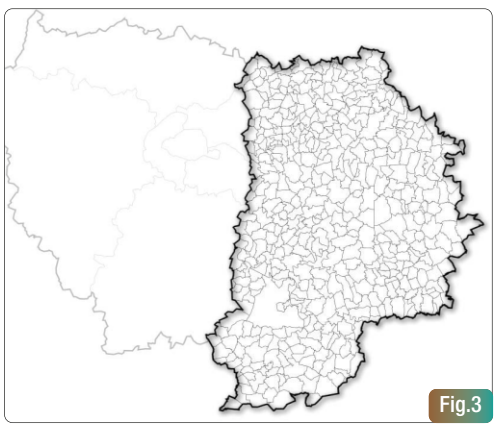


Fig.3

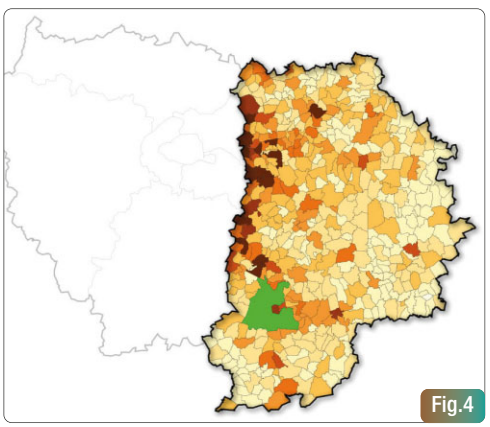


Fig.4

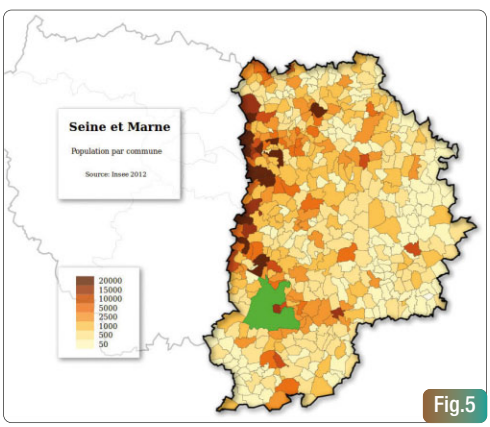


Fig.5

```
carte.append('rect')
     .attr('x', -20)
     .attr('y', 350)
     .attr('width', 270)
     .attr('height', 200)
     .style("filter", "url(#drop-shadow)")
     .style('fill', '#fff')
     .style('stroke', 'black')
     .style('stroke-width', '.3px')
     .style('opacity', '0.5');

carte.append('text')
     .attr('x', 5)
     .attr('y', 400)
     .text('Seine et Marne')
     .style('font-size', '26px')
     .style('font-weight', 'bold');
carte.append('text')
     .attr('x', 10)
     .attr('y', 450)
     .text('Population par commune')
     .style('font-size', '16px');
carte.append('text')
     .attr('x', 40)
     .attr('y', 500)
     .text('Source: Insee 2012')
     .style('font-size', '14px');
```

Un code similaire peut être mis en œuvre pour la légende en se fondant sur les valeurs des paliers de nombre d'habitants.

Nous affichons ces deux zones sur la gauche de la carte de la Seine-et-Marne pour la rendre plus agréable à regarder Fig.5.

Villes principales

Afin d'agrémenter un peu plus notre carte, nous pouvons également ajouter les localisations et noms des villes principales. Ces dernières vont être positionnées en se basant sur leurs coordonnées trouvées sur Internet. A partir de ces dernières, nous construisons un fichier CSV avec la structure suivante :

Colonne	Description
commune	Le nom de la commune
lat	La latitude de la commune
lon	La longitude de la commune

L'affichage de ces villes se réalise en deux phases: le positionnement d'un point et l'ajout d'une zone de texte affichant son nom.

Ajoutons des cercles pour chaque localisation de commune, comme décrit ci-dessous :

```
communesLabels.append('g').selectAll('circle')
     .data(communesPrincipales)
     .enter()
     .append("circle")
     .attr("cx", function (d) {
       return projection([d.lon, d.lat])[0];
     })
     .attr("cy", function (d) {
       return projection([d.lon, d.lat])[1];
     })
     .attr("r", "8px")
     .style("fill", "none")
     .style("stroke", "black")
     .style("stroke-width", "2");
```

Les rectangles et les zones de texte sont ajoutés de la même manière que pour les titre et légende. Nous ne les détaillerons donc pas à nouveau Fig.6.

Détail pour une commune

D3 permet d'ajouter un aspect dynamique à notre carte afin de visualiser les informations relatives à une commune en passant simplement la souris sur la zone correspondante. Nous devons à cet effet enregistrer des traitements sur les événements "mouseover" et "mouseout". Dans le premier cas, nous affichons l'info bulle, la positionnons à côté du curseur de la souris et affichons les bonnes informations. Dans le second cas, elle est cachée. Le code suivant met en œuvre ce mécanisme.

```
communes.selectAll('path')
        .data(contoursCommunes.features)
        .enter()
        .append('path')
        .attr('d', path)
        // ...
        .on('mouseover', function(d, i) {
          var c = d3.mouse(this);
          svg.select('#tooltip').style('display', 'inline')
             .attr("x", (c[0] + 15) + "px")
             .attr("y", (c[1] + 15) + "px");
          svg.select('#tooltip-text').style('display', 'inline')
             .attr("x", (c[0] + 25) + "px")
             .attr("y", (c[1] + 35) + "px");

          svg.select('#tooltip-text-id')
             .attr("x", (c[0] + 25) + "px")
             .text(d.properties.C_CAINSEE);
          svg.select('#tooltip-text-name')
             .attr("x", (c[0] + 25) + "px")
             .text(d.properties.B_BOIS === 'Oui' ?
                d.properties.L_BOIS : d.properties.L_CAB);
          svg.select('#tooltip-text-pop')
             .attr("x", (c[0] + 25) + "px")
             .text(d.properties.B_BOIS !== 'Oui' ?
                populationById[d.properties.C_CAINSEE] + ' habitants' :
                '');
        })
        .on('mouseout', function(d, i) {
          svg.select('#tooltip').style('display', 'none');
          svg.select('#tooltip-text').style('display', 'none');
        });
```

Une info bulle est désormais affichée dynamiquement en fonction de la position de la souris sur la carte Fig.7.

Affichage de données additionnelles

Il est également possible d'enrichir encore la carte en lui ajoutant par exemple les cours d'eau issus de OpenStreetMap. Il suffit simplement d'appliquer le contenu du fichier TopoJSON "waterways" comme nous l'avons fait pour les contours des communes :

```
var tracesRivieres = topojson.feature(rivieres,
          rivieres.objects['waterways-clipped']);
formes.selectAll('path').data(tracesRivieres.features)
      .enter().append("path")
      .attr("d", d3.geo.path().projection(projection))
      .style('fill', 'none')
      .style('stroke', '#2ECCFA')
      .style('stroke-width', '1.5px');
```

Le résultat final : Fig.8.

CONCLUSION

Comme nous l'avons montré, construire une carte pour un affichage en mode Web consiste en l'assemblage de différentes données afin de les mettre en corrélation les unes avec les autres. D3 se positionne en coordinateur afin de charger les données puis de les transcrire en données visualisables dans un navigateur, avec la technologie SVG dans notre cas. Nous avons utilisé des données statiques et ouvertes dans notre exemple, mais n'importe quelles données accessibles sur Internet pourraient être utilisées, notamment celles mises à disposition par des services RESTful.

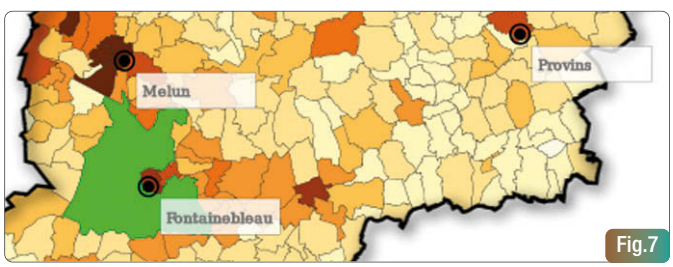


Fig.7

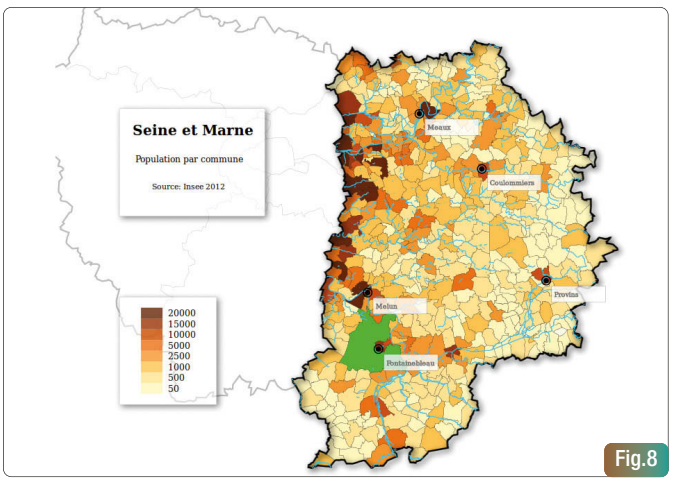


Fig.8

Les tests Android démystifiés

2e partie

Le développement d'applications Android a vu son écosystème évoluer et gagner en qualité. Le nombre de bibliothèques facilitant et accélérant les temps de développement a explosé grâce notamment à la communauté open source. Les différents outils à disposition des développeurs ont aussi grandement évolué, à l'image d'Android Studio, l'IDE officiel développé par Google.

Configuration

- Android Studio v1.0 ou plus
- Android SDK v22 ou plus
- Android SDK Tools v22.0.0 ou plus
- Android Support Repository v15 ou plus
- Plugin gradle: v1.1.0 ou plus


Florent Noël
Genymobile – Ingénieur Expert Android
@heb_dtc

Premier test

Dans un premier temps, testons que si aucun cookie n'a été sauvegardé alors le « UserCookieHandler » retourne une chaîne vide.

```
@Test
public void testThatIfNoCookieStoredReturnEmptyString() {
  //init
  userCookieHandler = new UserCookieHandler();

  //test
  String cookie = userCookieHanlder.getCookie();

  //success condition
  assertTrue(cookie.isEmpty());
}
```

Le test s'exécute ensuite soit directement depuis Android Studio, soit en ligne de commande.
En ligne de commande, à la racine du projet, avec le « wrapper » gradle il suffit de lancer la commande :

```
$ ./gradlew test
```

En cas de succès, BUILD SUCCESS s'affiche. En cas d'erreur, BUILD FAILED et un lien vers le rapport d'erreur est affiché.

De qui se « mock » t-on ?

Le précédent test était relativement simple car aucun élément extérieur n'était impliqué. Seulement, la plupart du temps, les classes utilisent et dépendent elles-mêmes d'autres objets. Or, ce n'est pas le comportement de ces objets que l'on souhaite tester mais le comportement de la classe qui les utilise. La solution serait donc de contrôler leurs comportements afin de tester si notre classe réagit correctement. On parle de « mocking » et grâce à la bibliothèque Mockito, il est possible de scripter intégralement le comportement d'objets.
Imaginons donc que la classe UserCookieHandler enregistre le cookie dans les « SharedPreferences ». Il faut donc modifier la classe afin qu'elle puisse utiliser les « SharedPreferences ». Une option serait de modifier le constructeur de la façon suivante :

```
class UserCookieHandler {

  private SharedPreferences sharedPreferences;

  public UserCookieHandler(Context context) {
    this.sharedPreferences = context.getSharedPreferences(..., ...) ;
  }
}
```

On passe le « Context » à notre classe et celle-ci se charge de récupérer les « SharedPreferences ». Cette approche bien que valide pose un problème pour mocker les « SharedPreferences ». En effet, impossible d'aller modifier une variable privée de l'objet, et, évidemment, ajouter un setter public n'est pas la bonne solution car cela exposerait le contenu de cette classe. Une technique fréquemment utilisée est celle de l'injection de dépendances. Ainsi, au lieu de laisser la classe allouer l'objet, la responsabilité est déléguée au créateur de cette classe.

```
class UserCookieHandler {

  private SharedPreferences sharedPreferences;

  public UserCookieHandler(SharedPreferences sharedPreferences) {
    this.sharedPreferences = sharedPreferences;
  }
}
```

De cette façon, rien n'est exposé au monde extérieur et surtout, il est maintenant possible de « mocker » l'objet « sharedPreference ».
Le test utilisant le mock s'écrit de la façon suivante :

```
class UserCookieHandlerTest {

@Mock
private SharedPreferences mockedPreferences;

@Before
public void init() {
MockitoAnnotations.initMocks(this);
}

@Test
public void testThatIfNoCookieStoredReturnEmptyString() {
  //init
    when(sharedPreferences.getString(UserCookieHandler.SP_COOKIE_KEY,""))
    .thenReturn("NO_COOKIE");

  userCookieHandler = new UserCookieHandler(mockedPreferences);

  //test
```

```
  String cookie = userCookieHanlder.getCookie();

  //success condition
  assertEquals("NO_COOKIE", cookie);
}
```

On notera l'utilisation d'une nouvelle annotation ***@Mock***.
Elle permet de marquer les mock. L'appel à :

```
MockitoAnnotations.initMocks(this);
```

assure que tous les mocks sont bien initialisés.
L'objet mocké peut ensuite être scripté et permet donc de vérifier le bon fonctionnement de la classe envers ses dépendances.

A travers ce simple exemple, la mise en place et la rédaction de tests unitaires a été expliquée.
Cette approche est relativement souple et permet de couvrir énormément de cas. Le recours à « Mockito » pour pouvoir mocker les objets rend leur scripting simple et verbeux. Les assertions peuvent être encore plus verbeuses et faciles à interpréter, avec l'aide de la bibliothèque AssertJ par exemple.
Autre point très important abordé ici, la nécessité de refondre le code afin de le rendre testable. Cette réalité est souvent dure à assimiler au début mais est absolument obligatoire.

Un peu de café ?

Pour faire des tests d'intégration, il existe de nombreux frameworks. Cet article s'attarde sur Espresso car il est officiellement supporté par Google et fait maintenant partie de la Android Support library.
Espresso permet de lancer les tests directement sur un terminal et d'instrumenter un parcours utilisateur.
Ici aussi, il faudra mettre à jour la liste des dépendances dans le build.gradle de l'application :

```
androidTestCompile 'com.android.support.test:rules:0.3'
androidTestCompile 'com.android.support.test:runner:0.3'
androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2'
```

De la même manière que pour les tests unitaires, les classes de tests ont leur propre dossier : app/src/androidTest
La dernière étape sera de configurer le « test runner ». L'Android Support Library propose l'AndroidJUnitRunner qui fera parfaitement l'affaire dans un premier temps. Pour indiquer qu'il faut utiliser celui-là, dans le « defaultConfig » du fichier « build.gradle », il suffit d'ajouter la ligne suivante :

```
android {
  defaultConfig {
    testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
  }
}
```

Il est aussi fortement recommandé de désactiver les animations sur les terminaux qui exécutent les tests. Les animations peuvent en effet provoquer des effets inattendus, voire même l'échec des tests.

Procédure :
- *Allez dans Paramètres > Options développeur,*
- *Désactivez Echelle animation fenêtre,*
- *Désactivez Echelle animation transitions,*
- *Désactivez Echelle durée animation.*

Premier test

Le premier test va vérifier la présence d'une « ToolBar » dans une « Activity ». De la même manière que pour les tests unitaires, une bonne façon d'organiser les tests consiste à les regrouper par classe. Ainsi, pour tester la classe « HomeActivity », il faut créer une classe « HomeActivityTest ».
Les classes de tests doivent hériter de « ActivityInstrumentationTestCase2<T> ». Attention à ne pas hériter de ActivityInstrumentationTestCase qui a été dépréciée.
Il faut aussi fournir un constructeur par défaut et bien penser à créer une méthode ***setUp()*** annotée de ***@Before*** afin de faire l'appel à ***super.setUp()*** et ***injectInstrumentation()***. Il est capital d'avoir cette méthode sinon les tests échoueront tous.
Pour ce qui est des tests, il suffit de les annoter avec l'annotation ***@Test***.

```
public class PreHomeActivityTest extends ActivityInstrumentationTestCase2<PreHomeActivity> {
  private PreHomeActivity preHomeActivity;

  public PreHomeActivityTest() {
    super(PreHomeActivity.class);
  }

  @Before
  public void setUp() {
  super.setup();
   injectInstrumentation(
     InstrumentationRegistry.getInstrumentation());
   preHomeActivity = getActivity();
  }

  @Test
  public void testToolBarIsShown() {
   onView(withId(R.id.toolbar)).check(matches(isDisplayed()));
  }
}
```

Espresso offre une API plutôt verbeuse ce qui permet d'écrire des tests faciles à comprendre. Un peu de la même manière que pour les tests unitaires, les tests d'intégration se décomposent en plusieurs étapes. Une première qui va créer le contexte souhaité et une deuxième qui vient vérifier ou non une assertion.
Les trois composants utilisés pour l'écriture d'un test sont les suivants :

ViewMatcher

La manipulation de la UI se fait de manière très naturelle pour un développeur Android puisqu'il est possible d'accéder aux éléments via leur ID. On parle alors de « ViewMatchers ». Il existe différentes alternatives pour cibler une vue, il est notamment possible de sélectionner un groupe de vues via les « Matchers » de la bibliothèque « Hamcrest » (intégrée à Espresso). Il est aussi possible de référencer une vue via ce qu'elle affiche, comme le texte d'un bouton par exemple.

Action

Une fois la vue sélectionnée, il est possible d'interagir avec; que ce soit pour simuler un clic, un scroll ou encore une entrée utilisateur. Il est aussi possible de chaîner des actions. On pourra par exemple faire un scroll puis un clic.

Assertion

Enfin les assertions servent à valider le comportement. Il est possible de les appliquer à la vue sélectionnée grâce à la méthode *check*. L'assertion la plus utilisée est sûrement le *match* qui vient vérifier l'état de la vue. Mais il est aussi tout à fait possible d'utiliser les méthodes *assert*.

QUELQUES EXEMPLES DE TESTS FONCTIONNELS

Test : le bouton Journal est-il bien présent ?

```
@Test
public void testButtonJournalIsShown() {
  final View decorView = homeActivity.getWindow().getDecorView();
  Button journalButton = (Button) homeActivity.findViewById(R.id.gotoJournalBtn);
  ViewAsserts.assertOnScreen(decorView, journalButton);
}
```

Le test se fait ici via l'utilisation de la classe ViewAssert au lieu de onViewCheck du test précédent.

Test : utilisation des Rules de JUnit

Une classe de test peut aussi ne pas hériter de « ActivityInstrumentationTestCase2<T> ». Il est alors possible d'avoir recours aux classes ActivityTestRule et ServiceTestRule afin d'instrumenter le lancement d'un Service ou d'une Activity. Leur utilisation se fait via l'annotation @Rule.
Cette approche réduit grandement la quantité de code à écrire. Si cette méthode est préférée au fait de faire hériter la classe de test de « ActivityInstrumentationTestCase2<T> », elle ne couvre cependant pas tous les cas et ne marchera pas avec un IntentService par exemple. Il faut donc jongler entre ces deux méthodes et appliquer celle qui convient le mieux à la situation.

```
public class MyActivityTest {
  @Rule
  public final ActivityTestRule<MyActivity> activityTestRule = new ActivityTestRule<>(My
Activity.class);

  @Test
  public void testActivityMethod() {
    Intent intent = new Intent();
    activityTestRule.launchActivity(intent);

    MyActivity activity = activityTestRule.getActivity();

    assertEquals(activity.isEmpty(), true);
  }
}
```

La rule qui lance l'Activity est déclenchée avant chaque test de la classe. Il est ensuite possible de récupérer l'instance et de tester ses méthodes publiques.

Test : monitorer le lancement d'une Activity

```
@RunWith(AndroidJUnit4.class)
public class UserJourneyTest extends ActivityInstrumentationTestCase2<PreHomeActivity> {

  private PreHomeActivity preHomeActivity;
  private Instrumentation.ActivityMonitor activityMonitor;
  private Instrumentation instrumentation;

  public UserJourneyTest() {
    super(PreHomeActivity.class);
  }

  @Before
  public void setUp() throws Exception {
    super.setUp();
    injectInstrumentation(InstrumentationRegistry.getInstrumentation());
    instrumentation = getInstrumentation();
    preHomeActivity = getActivity();
  }

  @Test
  public void testClickOnButtonJournalStartJournalActivity() {
    activityMonitor = new Instrumentation.ActivityMonitor(
      JournalActivity.class.getName(),null, false);

    instrumentation.addMonitor(activityMonitor);

    onView(withId(R.id.gotoJournalBtn)).perform(click());

    JournalActivity journalActivity = (JournalActivity)
      activityMonitor.waitForActivityWithTimeout(5000);

    assertNotNull("ReceiverActivity is null", journalActivity);
  }
}
```

Le lancement de l'Activity JournalActivity est monitoré grâce à la classe Instrumentation.ActivityMonitor.

Enfin, il est aussi possible d'utiliser Espresso pour écrire des tests unitaires instrumentés. Dans certains cas, l'utilisation de mock n'est pas suffisante. Il est alors possible d'écrire des tests instrumentés qui ne vont pas tester l'interface ou le parcours utilisateur mais des blocs métiers de l'application. Pour cela, il faudra plutôt que la classe de test hérite de ActivityUnitTestCase. C'est bien en combinant les différentes approches qu'il est possible de couvrir la totalité, ou presque, d'une application.

Pour aller plus loin

Les tests unitaires et les tests d'intégration étant en place, l'ultime étape est de faire en sorte d'avoir des métriques sur la couverture des tests grâce à un outil comme Jacoco.
Il est aussi possible d'aller encore plus loin dans la portée et l'automatisation des tests. Ainsi, dans certains cas de figure, il peut être intéressant de tester l'interaction de plusieurs applications entre elles. UiAutomator permet d'écrire ce genre de tests et offre une très grande liberté quant aux actions possibles qu'il est possible d'émuler. Avec une touche plus orientée QA, UiAutomator peut s'intégrer dans le processus de validation de l'application. ⊞

Liens

http://square.github.io/spoon/
http://developer.android.com/training/testing.html
http://robolectric.org/
http://developer.android.com/training/testing/ui-testing/uiautomator-testing.html
http://junit.sourceforge.net/javadoc/

iOS 9, Swift 2, qu'est-ce qui a changé ?

Le 16 septembre, Apple a rendu disponible iOS 9 pour les iPhones à partir du 4S et les iPad à partir du 2 ou du Mini. Comme à l'accoutumée, une majorité des utilisateurs ont fait la mise à jour et 3 semaines plus tard Apple annonçait fièrement 57% du parc déjà sous cette dernière mouture.

Pascal Batty,
expert iOS chez **SOAT**

Fig.1

Elle propose pourtant peu de nouvelles fonctionnalités majeures et certaines sont réservées aux derniers iPad comme le split-screen. C'est surtout, comme le nouvel OS X El Capitan, une mise-à-jour de consolidation avec des améliorations de la stabilité, des performances et de l'autonomie. Mais derrière cette façade relativement inchangée, quelles sont les nouveautés pour les développeurs d'applications ? **Fig.1.**

App Thinning

Comme Apple propose toujours des iPhones avec 16Go de stockage, la taille des applications est un souci et iOS 9 apporte 3 axes d'amélioration de ce côté **Fig.2**.

App Slicing répond à la prolifération d'assets de tailles différentes (@1x, @2x, @3x), au code compilé pour plusieurs architectures différentes (32 et 64 bits). Lorsque l'utilisateur télécharge l'application, l'App Store lui envoie une version avec seulement les assets et le code correspondants à son appareil.

On-Demand Resources est surtout utilisé pour les jeux et permet de stocker certaines ressources (des niveaux, textures, sons...) sur l'AppStore et de les télécharger depuis l'app au moment où elle en a besoin.

Bitcode est un nouveau code intermédiaire que le développeur envoie à Apple. Lorsque l'utilisateur télécharge depuis l'App Store, le code est automatiquement optimisé pour le CPU cible.

Nouveautés Autolayout

La plupart des développeurs entretiennent une relation d'amour-haine avec Autolayout, le moteur de résolution des contraintes qui permet de situer les contrôles à l'écran. Avec iOS 9, **Stackview** propose enfin de répartir ses éléments à la manière de la flexbox : une pile dynamique horizontale ou verticale **Fig.3**. Avec d'autres fonctions plus avancées comme les Layout Guides qui agissent comme des règles invisibles sur la vue, ou Layout Anchor qui facilite la création de contraintes dans le code source, Autolayout commence à devenir utilisable sans s'arracher les cheveux.

Safari View Controller et les Bloqueurs de Contenu

Si la *WebView* est très utile pour afficher du contenu Web généré par une application ou une navigation bien contrôlée comme un processus d'achat, certaines apps comme les clients Twitter, Facebook permettent d'accéder à des sites web arbitraires. Pour éviter de recoder un navigateur complet "in-app", iOS 9 propose maintenant Safari View Controller, qui s'exécute dans un processus séparé et intègre les fonctionnalités de Safari comme le remplissage automatique des formulaires et les extensions. Une autre nouveauté côté Web est l'autorisation des Bloqueurs de contenus : une app peut maintenant envoyer à Safari une liste d'éléments à masquer ou même à ne pas récupérer.

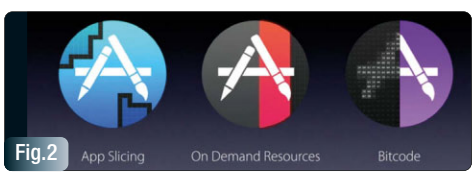


Fig.2

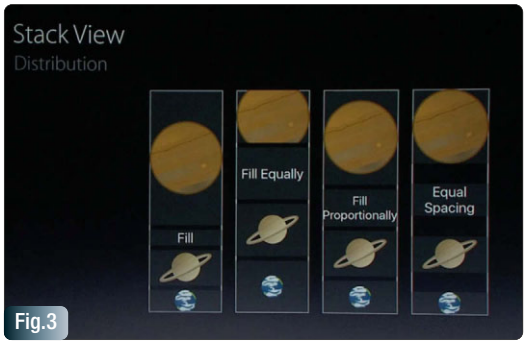


Fig.3

Spotlight

Le nouveau module de recherche d'iOS 9 peut à présent accéder à du contenu que les applications ont indexé, à l'aide des API CoreSpotlight pour l'indexation et NSUserActivity qui permet d'accéder directement au contenu à l'intérieur de l'application **Fig.4**. Mieux encore, certains contenus peuvent être indiqués comme "publics" et s'afficher dans le champ de recherche des utilisateurs qui n'ont pas l'application si suffisamment d'utilisateurs s'en servent.

GameKit

Après SpriteKit et Metal, Apple continue d'approvisionner les développeurs de jeux avec une panoplie de nouveaux outils, comme des shaders pour Metal, Model I/O qui permet de travailler au niveau système avec les modèles en 3D, mais aussi GameplayKit qui permet de modéliser l'IA et les mécaniques d'un jeu et ReplayKit qui permet d'enregistrer des replays facilement **Fig.5**.

App Transport Security

Afin de garantir une meilleure sûreté dans les accès réseau, une nouvelle couche de sécurité est posée sur les API standards comme NSURLSession : par défaut, les requêtes qui ne respectent pas les best practices (HTTP, TLS 1.2...) seront bloquées par le système. Le développeur peut définir des exceptions au niveau de l'application, pour le moment.

Intégration continue et tests UI

L'IDE Xcode 7 embarque aussi son lot d'améliorations comme une extension du système existant d'Intégration Continue Xcode Bots, mais également un affichage de la couverture de code et surtout un framework de Tests UI tout neuf en code natif (plutôt que JavaScript, précédemment). Un

Fig.4

bouton Record permet d'automatiser l'écriture du code de manipulation de l'UI en utilisant directement l'application **Fig.6**.

Swift 2

Swift 2 a été annoncé à la WWDC de juin avec notamment la promesse de le rendre Open Source pour Linux d'ici la fin de l'année. Apple mise beaucoup sur ce langage et semble vouloir en faire une référence et pour cause, cette mise à jour majeure à son premier anniversaire apporte quelques nouveautés bien attendues **Fig.7**.

Try/Catch et defer

Swift était sorti sans la moindre gestion des erreurs, ce qui avait surpris beaucoup de monde. Ce manque est maintenant comblé avec une syntaxe assez simple : try ne définit pas un bloc mais se place devant l'appel d'une méthode qui peut échouer, il faut alors soit gérer l'erreur dans un catch, soit signaler que la méthode peut échouer avec un throws.

```
func higherFunction() -> Bool {
  do {
    try thisCanFail()
  }
  catch {
    print("Oops !")
    return false
  }
  doSomethingAfter()
  return true
}
```

L'erreur n'est pas une classe mais un protocole ErrorType auquel il faut se conformer. Il est important de noter au passage que dans le bridge avec les fonctions Objective-C, la convention d'utiliser une référence vers NSError sera traduite avec ce nouveau système en Swift, propre ! Plutôt que de proposer un finally, Swift 2 apporte le mot-clé defer , que l'on peut mettre où l'on veut dans le scope et qui définit un bloc de code qui sera exécuté à la sortie, quoi qu'il arrive.

```
func higherFunction() throws -> Bool {
  defer { doSomethingAfterNoMatterWhat() }
  do {_
    try thisCanFail()
  }
  catch {
    print("Oops !")
    return false
  }
  try thisCanAlsoFail()
  doSomethi_ngAfter()
  return true
}
```

Fig.7

Guard

Ruby a montré que les synonymes (comme l'utilisation de if et unless) peut apporter de la clarté au code quand il est lu par des humains. Le mot-clé guard est un conditionnel comme if qui garantit que l'on sorte du scope s'il est rempli, par un return ou une erreur par exemple.

```
guard let name = json["name"] as? String else {
  return .None("missing name")
}
```

Ça peut être utile pour clarifier les cas limite en entrée de méthode, plutôt que d'imbriquer des if, surtout avec les tests des Optionals qui pouvaient donner choses assez sales.

Availability Checking

Même si la plupart des utilisateurs mettent docilement à jour leur appareil, ce n'est pas forcément le cas de tout le monde et il peut parfois être difficile de supporter les anciennes versions de l'OS tout en profitant des nouveautés alléchantes **Fig.8**. Xcode 7 observe maintenant les API utilisées par le code et signale si elles ne sont pas disponibles dans toutes les versions supportées par l'application. Il est alors possible de les isoler derrière une condition sur la version de l'utilisateur, ou bien de signaler qu'une fonction ou même une classe nécessite une version spécifique pour fonctionner.

Protocol Extensions

Swift permet, comme Objective-C, les protocoles (similaires aux Interfaces en Java ou en C#) et les extensions (possibilité de rajouter des fonctions à un objet). Swift 2 apporte la possibilité de déclarer des extensions de protocoles, en d'autres termes d'implémenter du code dans ces derniers. On se trouve alors dans un environnement qui permet une certaine forme d'héritage multiple ! Ça peut faire un peu mal à la tête mais les créateurs du langage y voient un nouveau paradigme : la Programmation Orientée Protocole.

Beaucoup de choses, en fait...

Ce n'est pas tout : Swift 2 intègre de nouvelles façons d'utiliser le Pattern Matching et beaucoup de sucre syntaxique, 3D Touch implique également de nouvelles API, les Storyboards sont plus faciles à utiliser... Une mise à jour qui ne semble pas changer énormément de choses pour les utilisateurs fourmille en fait de nouveautés juteuses. Cette année 2015 se montre finalement très riche du côté des développeurs de l'écosystème Apple, avec deux nouveaux OS pour lesquelles créer de nouveaux types d'applications : watchOS 2 et tvOS. ⊞

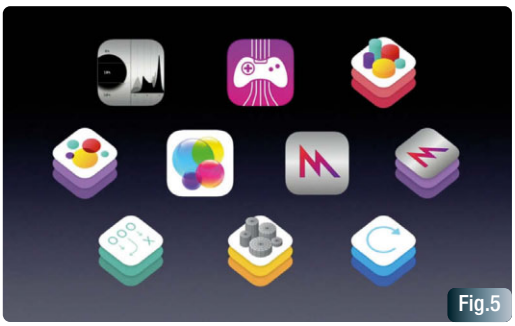


Fig.5

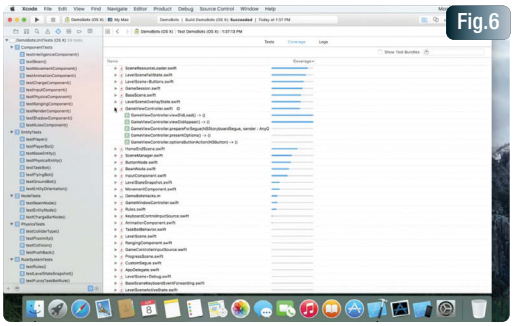


Fig.6

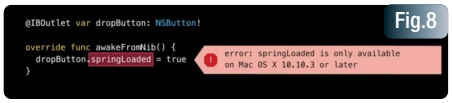


Fig.8

Les nouvelles architectures logicielles

1ère partie

L'informatique repose sur des architectures matérielles et logicielles. Je ne vous apprends rien. Certaines existent depuis quasiment les origines de l'informatique « moderne », dans les années 1970, d'autres sont bien plus récentes. On parle beaucoup du Cloud Computing, mais le Cloud n'est finalement qu'une évolution d'architectures utilisées depuis de nombreuses années.

Les architectures évoluent tout naturellement avec les évolutions techniques, les nouveaux langages, les nouvelles technologies. Ces évolutions ont un impact direct sur les applications et donc sur vous, les développeurs. Ces changements sont parfois masqués, car vous n'êtes pas toujours au contact direct.

Programmez ! vous propose un dossier en plusieurs parties pour explorer quelques architectures. Nous aborderons, en vrac : le Cloud Computing, les architectures dites lambda, les microservices, les conteneurs.

Dans cette 1ere partie : nous parlerons de microservices et de Java.

La rédaction.

Les architectures microservices

L'architecture Microservices a fait couler beaucoup d'encre dernièrement : il est temps de faire le point.

Jérôme Doucet

Romain Niveau

Consultants Xebia

L'idée derrière ce terme est d'isoler chaque fonctionnalité dans un composant qui lui est propre. Les différents services découlant de ce découpage communiquent au travers du réseau plutôt que par des appels de fonctions dans un processus. Une application microservices représente alors un système constitué d'un certain nombre de petites unités indépendantes qui collaborent entre elles, ayant chacune un cycle de vie et une responsabilité propre.
Cette architecture répond à des problèmes récurrents que posent les architectures monolithiques traditionnelles.

LES CONCEPTS

Les Architectures monolithiques et leurs problèmes

L'architecture monolithique est l'une des plus répandues dans les projets informatiques.
Plusieurs raisons expliquent cela. Au début d'un projet, quand celui-ci est encore de petite taille, elle reste très simple à comprendre favorisant la maintenabilité et la testabilité. Le déploiement, facile et rapide à mettre en place, permet de se concentrer sur les fonctionnalités plutôt que sur l'infrastructure. Il permet également de faire de l'intégration continue à peu de frais. Cependant, au cours de l'évolution d'un projet, cette architecture montre fréquemment ses limites.

Maintenabilité du code

Si en début de projet, la simplicité du modèle monolithique favorise grandement la compréhension du code, ceci devient de moins en moins vrai au fur et à mesure que le volume de code augmente, et ce, même avec une organisation rigoureuse de ce dernier. Un nouveau venu mettra davantage de temps à être pleinement opérationnel.
Parce qu'il n'y a pas de frontières fortes entre les modules d'une application monolithique, elles auront tendance à s'effacer, affaiblissant la modularité, complexifiant les relations de dépendance. Le code devient de moins en moins lisible et testable, la productivité et la qualité baisse et enfin la dette technique s'accumule.

Un autre effet pervers de l'architecture monolithique est de rendre une application dépendante à certaines technologies, obligatoirement compatibles entre elles. Choisir Java limitera les évolutions futures aux technologies compatible avec la JVM, C# aux technologies Microsoft, etc. Si un framework utilisé par l'application devient obsolète, la migration vers un nouveau peut nécessiter la réécriture complète de l'application avec les risques que cela comporte.

Déploiement en continu

Une application monolithique semble simple à déployer. Puisque toute l'application est destinée à s'exécuter sur un même serveur, celle-ci est très souvent contenue dans une archive déployée sur celui-ci. Une usine logicielle pourra rapidement et facilement être configurée avec les taches de déploiement adéquates. Cependant, déployer toute l'application, quelles que soient les modifications réellement effectuées, pose divers problèmes. Premièrement, le temps de déploiement s'allonge, tendant à limiter leur nombre. En effet, la taille de l'application augmentant, le container prend plus de temps pour effectuer les démarrages applicatifs.
En conséquence, le déploiement continu

devient plus complexe, ce qui engendre une augmentation des risques associés à chaque déploiement et une perte de productivité.

Scalabilité

Lorsque la charge d'une application augmente, la scalabilité de l'application devient un point critique. Une application monolithique n'ayant qu'une capacité assez limitée (car celle-ci ne se fait que dans une dimension), on ne peut avoir que des copies de l'application entière. Or les différents composants de l'application n'ont vraisemblablement pas les mêmes besoins en ressource. Certains vont faire un usage intensif du processeur, d'autres de la mémoire, d'autres des entrées / sorties. Scaler une application monolithique peut entraîner une augmentation de la consommation des ressources conséquentes de manière inutile.

Les principes de l'architecture microservices

Une nouvelle échelle

Ce qui est le plus visible de prime abord est le changement d'échelle qui s'opère avec les microservices. L'application devient un système d'informations, dont le microservice constitue l'unité élémentaire. Ces unités sont strictement indépendantes les unes des autres. Elles ont chacune leur propre base de code de donnée et pile technologique. Chaque service peut ainsi utiliser les technologies les plus adaptées à son besoin. Il devient aussi possible d'allouer les ressources souhaitées à chaque type de service. Il est également envisageable de choisir un nombre d'instances différent par service, améliorant grandement l'efficacité de la scalabilité. Un service devant supporter une forte charge possède plus d'instances d'exécutions qu'un service faiblement sollicité. Selon le même principe, les données d'un service peuvent être mises à jour, migrées, traitées sans risquer d'impacter les autres fonctionnalités du système. Dans une architecture microservices, chaque service est différent évoluant selon son propre rythme.

Évolutivité et automatisation

Lorsqu'une fonctionnalité précise du SI doit être modifiée, l'évolution en question ne porte que sur le code d'un service en particulier. Les autres services ne seront pas impactés. Ainsi, la modification du code ou des données d'un service provoque uniquement la mise à jour et le redéploiement de celui-ci car il s'exécute dans un processus isolé. Il n'y a aucune obligation d'embarquer des mises à jour d'autres fonctionnalités développées en parallèle, celles-ci impactant d'autres services dont le code est séparé. De la même façon, la présence d'une anomalie sur une fonctionnalité ne bloque pas l'évolution ou le déploiement d'une autre. Il est possible, grâce à ces caractéristiques, d'envisager de prototyper, déployer et tester une nouvelle fonctionnalité sans remettre en cause l'ensemble des services. Dans cette architecture, l'échelle de la modification est donc, non pas un ensemble de fonctionnalités assemblées dans une "application", mais la fonctionnalité, portée par le service.

Services et communication

Pour que le système fonctionne correctement, il est donc nécessaire que les services communiquent entre eux. Le vecteur de cette communication a pour unique rôle d'assurer une transmission fiable des messages. Le système suit la règle suivante : *"Smart end point, dumb pipe"*. L'intelligence du système est contenue dans les services, pas dans le vecteur. Les deux modes de communication les plus courants sont celles de type REST et bus. Lorsqu'un service dépend de messages fournis par d'autres services en amont, la communication est généralement asynchrone et une architecture microservices va tendre vers une architecture réactive, à base d'évènements déclenchés et écoutés par les services.

Tolérance et monitoring

Une des grandes forces des architectures microservices est sa robustesse et sa tolérance aux pannes. En effet, plutôt que de se protéger à tout prix de ce qui pourrait poser problème dans le système, l'approche adoptée par les microservices est l'adaptation. Un service est conçu dans l'optique que le reste du système puisse être disponible, qu'une donnée nécessaire puisse être absente ou au contraire qu'un message puisse contenir des données superflues. Le service est alors capable de fonctionner à la fois en mode nominal et en mode dégradé. De plus, ses interfaces, suffisamment souples, supporteront les écarts.
Malgré cette tolérance, le système doit être surveillé en permanence. Les pannes peuvent compromettre le fonctionnement du système global. Ainsi, il devient nécessaire d'être capable de détecter rapidement une panne pour pouvoir intervenir. Il est également important de surveiller le système et les services d'un point de vue fonctionnel. Son bon fonctionnement passe par la surveillance d'indicateurs métiers : transactions validées, en erreur, quantité de commandes, envois de messages en erreur, nombre d'inscriptions, etc. Tous ces indicateurs métiers sont les premiers révélateurs d'un problème.

LA MISE EN OEUVRE

Quels outils ?

Cloud

Les architectures microservices ont besoin d'une infrastructure souple et robuste à la fois. Un nouveau serveur doit pouvoir être provisionné en quelques minutes pour permettre à un service de monter en charge rapidement. Il doit également pouvoir être décommissionné facilement si celui-ci n'est plus utile. Le Cloud représente le candidat privilégié pour répondre à ces besoins. Des solutions de Cloud public comme AWS d'Amazon ou Azure de Microsoft permettent de réaliser ces opérations pour un coût moindre par rapport à une infrastructure privée. Ils proposent de nombreux services allant du provisionning de CDN aux solutions d'API management. Du coté des Clouds privés, OpenStack se place comme la solution permettant de construire son propre Cloud. La solution est sponsorisée par des sociétés comme Google ou encore Red Hat.

Conteneurs

Issue de travaux anciens sur le système d'exploitation Unix (les C-groups), la conteneurisation arrive aujourd'hui à maturité, le projet Docker en est l'exemple le plus significatif. Cette technique consiste à isoler l'utilisation des ressources de type processeur, mémoire et disque par application sur une même machine. L'utilisation de ces technologies dans une architecture microservices offre un avantage indéniable. Chaque service correspond à une image, livrée sur un «catalogue» d'entreprise. Celle-ci peut être construite et mise à disposition directement par l'usine logicielle. Ainsi, il n'est plus nécessaire de s'adapter à des principes de déploiement propre à chaque écosystème logiciel (JEE, Play, Vert.x, Node.js, Python, etc.). Le déploiement est uniformisé et simplifié : il s'agit de déployer un conteneur par microservices. La conteneurisation facilite également les tests d'intégration, qu'ils soient réalisés sur le poste du développeur ou sur un serveur dédié. En effet, il est beaucoup plus facile de recréer de toutes pièces des environnements à base de conteneurs que de créer une infrastructure physique supportant un monolithe et ses dépendances, à l'image de la production. La boucle de feedback, ainsi accélérée, est portée directement par l'usine logicielle et ne nécessite plus de déploiement sur un environnement identique à la production décorrélé du cycle de développement.
On touche ici à une problématique à la limite

des mondes Dev et Ops. Le choix de cette technologie, impactant toutes les couches du SI, doit se faire de manière concertée.

Supervision

Les bases de données de type Time Series permettent de stocker, avec une grande fiabilité, des métriques à intervalles réguliers. Les équipes opérationnelles ont l'habitude de travailler avec ce type de bases qui permettent de récolter les informations minimales utiles à la détection des pannes. Il s'agit en général des métriques bien connues comme le CPU, la mémoire, l'espace disque, les entrées / sorties, etc. Les développeurs y portent un intérêt croissant, en particulier pour stocker des métriques applicatives, techniques voire métiers. Graphite, très populaire ces dernières années, a atteint ses limites en termes de scalabilité. On lui préférera donc des projets plus récents, embarquant nativement la notion de clustering tels que InfluxDb, entièrement compatible avec Graphite, talonné de près par OpenTSDB. Le stockage des données est important, l'exploitation et la visualisation le sont tout autant. Aucune des bases de données nommées ci-dessus ne propose de système d'alerte. En revanche, elles s'intègrent très facilement avec des outils classiques d'exploitation, comme Nagios, Shinken, Icinga ou Sensu. Les outils de dashboarding basés sur Graphite sont nombreux et leurs évolutions récentes permettent de s'intégrer facilement avec InfluxDb. Le projet le plus populaire est Grafana.

Framework

Les services des architectures microservices sont par définition légers en code et en fonctionnalités. Il doit donc en être de même pour les frameworks utilisés au sein de ces services. Par exemple, Vert.x apporte plusieurs briques nécessaires dans la communication entre services comme un bus de messages ou un serveur de websocket. Léger, son coté polyglotte permet de choisir le bon langage pour le bon service tout en ayant la même base, ce qui est un plus en termes de cohérence pour le système. Spring Boot permet quant à lui de développer rapidement des services grâce à ses différents profils d'application prêts à l'emploi. L'objectif de Spring Boot est de pouvoir déployer facilement des applications standalone embarquant elles-mêmes le serveur choisi (tomcat ou jetty), le tout fournissant par défaut des métriques permettant un suivi de l'application en production. Enfin, sans en être un framework, node.js représente un exemple type d'outil utilisé dans une architecture microservices. Sa légèreté, sa rapidité d'exécution couplée à la rapidité du développement lié au langage Javascript en font un très bon candidat pour des microservices. Le modèle i/o non bloquant de node.js lui permet de tenir particulièrement bien la charge.

Quelle organisation ?

La mise en place d'une architecture microservices dans des équipes implique une certaine réorganisation. Une architecture microservices induit des micro-équipes, en interaction constante, connaissant parfaitement le métier et la technique des quelques services à leur charge. Les services correspondent à une fonctionnalité issue d'un besoin métier autour duquel s'organise une équipe (Feature team) qui a la maîtrise du produit.
De la multiplicité des services découle le besoin d'automatisation des déploiements. Il n'est plus envisageable lorsque les services se comptent en dizaines de les déployer manuellement. En conséquence, les équipes doivent accueillir de nouveaux processus d'automatisation dans lesquels l'outil occupe une place importante. Le mouvement DevOps est la continuité logique de la mise en place d'une architecture microservices, permettant réactivité et souplesse en production.
Enfin, du point de vue des évolutions, il est important de garder un contrôle et une vision d'ensemble, grâce, par exemple, à une cartographie de son SI. Celle-ci permet de ne pas développer des fonctionnalités identiques et de connaître le spectre technologique dont est composé son SI. Ainsi, on veille à appliquer les correctifs de sécurité sur l'ensemble de son parc technologique et ne pas s'exposer aux attaques dirigées vers des versions vulnérables.

LES LIMITES

De nouveaux problèmes ?

Les microservices proposent des solutions à des problématiques connues et héritées d'architectures orientées services déjà en place. Néanmoins, toute nouvelle solution apporte des problèmes qu'il est possible d'identifier afin de s'en prémunir.

Taille des services

Le plus grand principe des microservices concerne la taille de ces services. Celle-ci n'est pas régie par des règles précises du fait de l'absence (souhaitable) de norme ou de spécification. Plusieurs pièges sont toutefois à éviter. Il est fréquent lors du développement de services de vouloir regrouper plusieurs "petits" services en un seul "macroservice" afin de faciliter la communication entre celui-ci et le reste du SI. Ceci va regrouper la complexité des services en un seul qui va de facto, devenir plus difficilement maintenable. Un autre écueil est la tentation du "nanoservice". Transformer chaque méthode d'une application monolithique en microservice générera une quantité importante de services. La communication de ses services n'en sera que plus complexe pour un apport limité en termes de souplesse. Il est donc important de trouver un juste milieu entre des services imposants et des services trop petits. Une règle communément admise est: une personne qui découvre un service doit pouvoir en comprendre le code et le fonctionnel en une journée.

Manque d'automatisation

La philosophie des microservices est de pouvoir ajouter rapidement et simplement des services à un système. Si le temps de déploiement ou le redémarrage d'un service nécessite une intervention humaine coûteuse en temps, alors le système se confrontera à un problème. Pour l'éviter et permettre un système évolutif et résilient, il faut automatiser au maximum les déploiements d'applications mais également l'ajout de nouveaux serveurs. Pour cela, le Cloud permet de bénéficier de capacités machines quasiment illimitées et des outils, comme Docker, facilitent l'automatisation des déploiements et la réplication des services.

Système de supervision défaillant

La multiplicité des services entraîne une complexification de la supervision du système. Que la communication entre services se passe par bus de messages ou par APIs, cette supervision doit être automatisée au maximum. En effet, le nombre de services croît régulièrement au fur et à mesure de la vie du système. Il devient vite impossible pour un humain de suivre tous les services manuellement. Pour pallier ce problème, chaque service doit fournir ses propres métriques au système de monitoring. Elles doivent être techniques mais également business afin de pouvoir mesurer en temps réel l'apport métier d'un service. Des règles d'alertes automatiques doivent être mises en place afin de savoir lorsqu'un service commence à ne plus fonctionner correctement. L'idéal est de disposer d'un système autonome capable de relancer un serveur défaillant de manière automatique, toujours sans intervention humaine.
Un autre problème lié à cette multitude de services est l'exploitation des fichiers de logs. Pour un service unique déployé sur plusieurs

serveurs, celle-ci peut s'avérer être fastidieuse. Une bonne solution consiste à centraliser les logs des services au même endroit, dans le système lui-même avec des solutions comme Flume ou ELK ou bien grâce à des services SaaS comme papertrail ou logmatic.

Profusion des technologies

Même si les microservices permettent l'utilisation de différents langages, il est important de veiller à ne pas multiplier les langages inutilement. Cette profusion de technique rendra le passage d'un service à un autre compliqué pour les développeurs et complexifiera le système inutilement.
Le même constat peut être fait avec les bases de données. Il est possible, et même conseillé, de choisir un type de base de données correspondant à chaque service. Il faut toutefois éviter de multiplier le nombre de bases différentes. Garder un type de base relationnelle et un type de base NoSQL peut être un bon compromis.

Sur le chemin des microservices

Avec l'émergence des architectures microservices, beaucoup de projets se sont lancés dans l'aventure sans toujours rencontrer le succès escompté. En effet, comme nous l'avons vu précédemment, les microservices répondent très bien aux problématiques de charge et de scalabilité mais peuvent compliquer le système dans son ensemble.
Fort de ce constat, Marin Fowler, un des précurseurs de ce type d'architecture propose deux concepts permettant un cheminement vers les microservices : Monolith First et Sacrificial Architecture.

Monolith First

Deux observations s'imposent aux yeux de Martin Folwer :

- La plupart des succès de ces projets viennent d'un monolithe devenu trop gros et qui a éclaté en microservices;
- La plupart des projets ayant démarré directement en microservices ont connu de sérieux problèmes en cours de route.

Plusieurs raisons expliquent ces observations. Les microservices sont efficaces si leurs domaines fonctionnels sont bien isolés. Au début d'un projet, il est parfois complexe de délimiter précisément chaque service. Ceci va entraîner plusieurs phases de refactoring qu'il est bien plus facile d'entreprendre dans un monolithe que dans plusieurs services communiquant via des APIs ou un bus.
Lors de l'initialisation d'un projet, une architecture microservices sera plus longue à mettre en place. Pour une startup ou un projet sans visibilité forte du marché potentiel, ce coût de démarrage peut être un problème quand le but est d'arriver rapidement à un produit fini.

Sacrificial Architecture

Le deuxième concept proposé menant sur le chemin des microservices est la Sacrificial Architecture. Ce concept part du constat suivant : le code écrit aujourd'hui pour un projet ne sera plus adapté dans deux ans.
En effet, il est très difficile voire impossible de savoir comment va évoluer un projet. Le succès sera peut être immédiat nécessitant une évolution rapide du code. A contrario, le projet ne rencontrera peut être pas son public et un investissement trop lourd dès le départ aura des conséquences néfastes pour l'entreprise. Au lieu de prévoir l'évolution à long terme de l'architecture du projet, il est préférable d'initier une architecture jetable permettant de sortir rapidement un produit. Ensuite seulement, en fonction de l'évolution du projet, il sera possible de poser une architecture plus solide, absorbant mieux la charge et permettant une évolutivité plus simple. Le rapprochement avec le concept de monotith first est évident. Suivant le projet, il peut être préférable de partir sur une architecture simple et rapidement opérationnelle afin de sortir le plus tôt possible un produit puis d'adapter (voire même de jeter) l'architecture pour évoluer vers un système plus maintenable et robuste.

CONCLUSION

Les architectures orientées services ont piloté la mise en place des SI de ces 10 dernières années. Le constat majoritaire est une capacité à évoluer qui diminue avec le temps du fait de la difficulté d'isoler les évolutions dans le système. En effet, les applications dont il est composé sont en majorité des monolithes qui réunissent un assemblage hétérogène de fonctionnalités de plus en plus nombreuses avec le temps. À ces problématiques, les microservices proposent une réponse simple : diminuer l'échelle de l'architecture. Cette échelle traduit une isolation nécessaire des fonctionnalités à différents niveaux : code, données, déploiement et exécution. Cette réduction de la granularité a des revers et exige un certain investissement au niveau technologique, méthodologique (Craftsmanship, DevOps) mais aussi et surtout au niveau organisationnel (Agilité, Feature Teams). L'émergence de nouveaux outils facilite également le passage aux microservices. Le Cloud ou les technologies de conteneurisation sont les instruments idéaux pour leur mise en place.
Toutefois, les microservices ne constituent pas non plus une recette universelle. Un système peu complexe sans problématique de charge ou dont le rythme d'évolution est faible ne bénéficiera pas de l'approche microservices. Ils ne garantissent pas une simplification du système global mais permettent la simplification de son évolution. Certaines briques du système ne sont en effet pas constamment bousculées par des besoins d'évolution. On peut donc tout à fait envisager de faire cohabiter le cœur monolithique du système, plus constant, avec en périphérie des briques conçues en microservices pour les besoins les plus mouvants.

L'art de l'architecture en Java

On va mettre l'accent sur les étapes à suivre pour aboutir à une bonne architecture, solide, robuste et maintenable. Pour commencer, on va voir comment bien choisir les technologies. Ensuite on va examiner l'architecture en étudiant les différents outils à utiliser dans un projet. Puis on va voir les techniques et les stratégies à mettre en place pour bien développer. Enfin on va voir comment maintenir le code à travers les tests unitaires et les tests d'intégrité.

Bouhanef Hamdi
Développeur full-stack chez **sfeir**
https://fr.linkedin.com/in/bouhanef-hamdi-62386051

Dridi Manel
Freelance web
https://www.linkedin.com/in/manel-dridi-ep-bouhanef-908316105

C'est quoi une architecture logicielle

Il existe plusieurs définitions pour dire ce qu'est une architecture. Une architecture peut être vue comme un ensemble des techniques, règles et paradigmes qui devraient être mis en œuvre pour garantir le bon déroulement des processus du développement d'un logiciel. On peut aussi le voir comme la décomposition physique du système et la communication entre ses composants, notamment dans les architectures employées par les équipes de l'infrastructure. Cela peut être encore la décomposition logique du besoin en plusieurs sous-composants, qui sont indépendants (pour la maintenabilité) et qui communiquent aussi entre eux. Nous allons dans cet article mettre la lumière sur la première et la troisième architecture.

Diviser pour mieux régner

Tout d'abord, il est préférable d'étudier tous les besoins du projet (même les besoins prévisionnels) et d'essayer de les diviser en sous-modules (surtout quand le projet est assez grand). Ces composants doivent pouvoir exister indépendamment des autres composants, ils ont leur propre cycle de vie et exposent une API pour communiquer, cette API peut être soit du REST, SOAP, RMI, RPC... Quelle que soit la norme utilisée, le module doit pouvoir communiquer avec l'extérieur ou avec les autres modules.

Le médiateur

Ce composant est facultatif, cela dépend du choix fait au début. Les modules peuvent avoir un cycle de vie indépendant et une API indépendante (dans le cas des APIs Google par exemple), ou peuvent être gérés au sein d'un conteneur, qui, lui, va gérer le cycle de vie du module ou composant; il gérera aussi la communication entre les composants (dans le cas de l'IDE Eclipse par exemple, les plug-ins sont les composants ou les modules et le core d'Eclipse est le médiateur).

CHOIX TECHNOLOGIQUE

Avec l'explosion des technologies et des frameworks, il devient de plus en plus difficile de choisir la ou les technologies à mettre en œuvre pour réussir le projet.

Les types d'architectures

Il existe plusieurs types d'architectures logicielles, suivant votre besoin, vous pouvez choisir l'une ou l'autre.

Les architectures 1 tiers

Vu la révolution technologique à laquelle on fait face, cette architecture a tendance à disparaître, on l'appelle aussi application stand alone. Ces applications en général ne communiquent avec aucun composant sur le réseau et sont contenues dans une seule unité physique (ordinateur ou terminal). Par exemple les applications de ligne de commande ou une application sur votre PC qui ne fait aucune interaction avec l'extérieur.

Les architectures deux tiers (Client-Serveur) Fig.1.

Les architectures deux tiers sont aussi appelées des architectures client-serveur. Dans ce type d'architecture, on a deux composants qui sont le client, c'est celui qui fait la demande, et le serveur qui la traitera et renverra une réponse; par exemple cette architecture est utilisée pour communiquer entre une application et la base de données.

Les architectures n-tiers

Dans ce type d'architecture, on peut avoir n modules (aussi appelés composants ou services) qui sont séparés d'une manière logique (séparer les traitements et les données dans différents packages, mais dans le même ordinateur) ou physiquement (mettre les services dans des machines différentes). L'avantage avec ce type d'architecture, c'est qu'il est ouvert à tous les systèmes. Quand on conçoit que notre système est de type n-tiers, les services sont ouverts à tout type de technologies (REST, SOAP...) Fig.2.

Couche Présentation

Cette couche dépend des technologies utilisées, et dépend de la portabilité de l'application. On va voir dans cette section les différents types de terminaux et les différentes structures à mettre en œuvre pour cette couche.

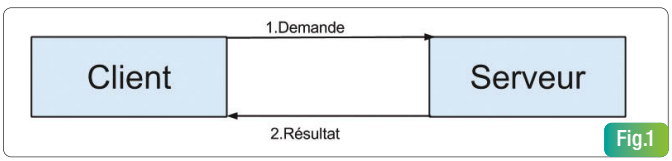


Fig.1

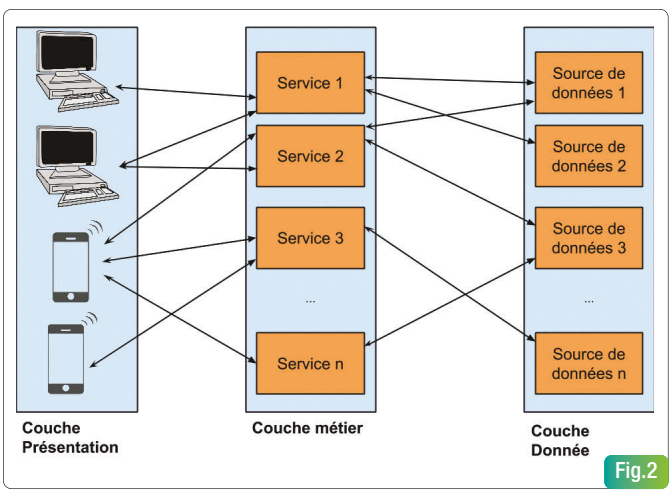


Fig.2

HTML généré côté serveur

Dans ce type d'architecture, la couche présentation est spécifique à un seul type de présentation qui est le Web, en plus le HTML est généré côté serveur et envoyé au client. Plusieurs technologies et frameworks permettant de créer ce type d'architecture :

- **JSP (Java server page), Servlets et JSTL (Java server page Standard Tag Library) :** Les JSP, les servlets et les JSTL sont les composants Web de la spécification Java EE qui permettent d'ajouter une touche dynamique côté serveur pour mettre à jour une ressource avant de l'envoyer au client.
- **JSF (Java server face) :** JSF est un framework spécifié par Oracle, qui permet d'implémenter le patron de conception MVC. Avec JSF, la gestion des objets est transparente en plus à partir de sa version 2. JSF a beaucoup facilité son utilisation avec les annotations.
- **Struts :** C'est un concurrent de JSF, chacun a ses avantages et ses inconvénients.

HTML + AJAX

Après l'introduction de la technologie AJAX (Asynchronous JavaScript and XML), la communication avec le serveur est devenue plus facile et plus rapide. Avec cette technologie, le navigateur n'aura pas à recharger tout le HTML à chaque demande. Le principe est d'envoyer en arrière-plan des demandes au serveur Web, puis de modifier le contenu de la page actuellement affichée par le navigateur Web en fonction du résultat reçu du serveur Fig.3. Il y a des frameworks qui permettent de combiner les deux (JSF et STRUTS), puisque le HTML est généré par le serveur et contient le mécanisme AJAX.

Génération JavaScript à la compilation

Un troisième type de framework permettant de générer le code source du front (JavaScript) avant le déploiement de l'application, dans le processus de compilation. Un exemple très connu de ces frameworks : GWT (Google Web tool-kit) qui est un framework permettant d'écrire le code de l'IHM en java, comparable à swing, et génère le JavaScript correspondant, et cela avant le runtime.

Couche Métier

Dans cette partie, on va voir en détail comment choisir les technologies pour la partie serveur, puis la différence entre chaque technologie.

Injection de dépendance et inversion de contrôle

L'inversion de contrôle est un mécanisme puissant permettant de reporter l'initialisation des objets et l'injection de leurs valeurs pour plus tard; il est utilisé par la plupart des frameworks pour prendre le contrôle sur les objets de l'application sans que l'application soit à l'origine de cette demande. Il y a plusieurs frameworks permettant de gérer l'inversion de contrôle et l'injection de dépendance, voici les plus connus :

- **Guice :** Google Guice est un framework créé par Google, permettant d'introduire un mécanisme d'injection de dépendance.
- **Spring core :** Spring est un framework qui propose des solutions pour la plupart des besoins (batch, sécurité, Web service, injection de dépendance, inversion de contrôle, dynamique proxy.) avec un mécanisme robuste dont l'injection de dépendance. La configuration est facile à mettre en place, surtout après la version 3 de Spring, l'utilisation des annotations est devenue possible.

Web service REST

Plusieurs frameworks sont mis en place pour pouvoir faciliter et normaliser l'utilisation des APIs (REST). En voici quelques-uns :

- **Spring MVC :** Spring MVC est le module de Spring qui permet de gérer une partie du front-end ou de la couche présentation, mais aussi permet d'exposer des API REST avec des annotations et d'une manière transparente.
- **Jersey (JAX-RS) :** Il permet de faciliter le développement des services Web en implémentant les JSR 311 et JSR 339 d'Oracle.
- **Google Endpoint :** Cette technologie est créée par Google, et utilisée exclusivement si l'application est dans google Appengine. Elle permet de créer les APIs et de gérer sa sécurité.

Web service SOAP

Les Web services Soap sont des services permettant de communiquer entre les plateformes hétérogènes comme les Web services REST, à la différence de ce type de Web services qui sont plus sécurisés nativement, et sont plus structurés (la structure d'échange est définie dès le début par les WSDL (Web Service Description Language).

- **JAX-WS :** Cet ensemble d'APIs présente plusieurs facilités pour implémenter le service et le client.
- **Spring Web Services :** C'est un des modules Spring qui permet de créer, gérer et déployer facilement des services Web de type Soap.

Couche données

Dans cette partie on va voir plus en détail la partie données, et comment gérer, modifier et supprimer les données. On va également voir quelques astuces pour augmenter la performance d'une application.

Les bases de données relationnelles

Les bases de données relationnelles sont les bases de données dont les objets élémentaires (ou tables) sont en relation.

- **Oracle :** Oracle est l'un des gestionnaires de bases de données les plus utilisés au monde, notamment pour les secteurs délicats comme le secteur de la finance, des banques... Son avantage est qu'il peut supporter une très grande volumétrie de données, et gérer efficacement la sécurité.
- **Postgresql :** base de données open source et gratuite.
- **MySql :** Cette base de données peut gérer efficacement les données.

Les bases de données No-SQL

Ces bases de données, à l'inverse des bases de données relationnelles, ne se fondent pas sur les relations entre les objets. Utilisées par la plupart des géants comme Google, Facebook... Il y a plusieurs représentations, suivant les besoins, des bases NoSQL :

- **A base de clés-valeurs :** C'est la plus simple à mettre en œuvre, mais inefficace si la manipulation majeure de la base est effectuée sur des parties des valeurs. Exemple Oracle NoSQL, CouchDB

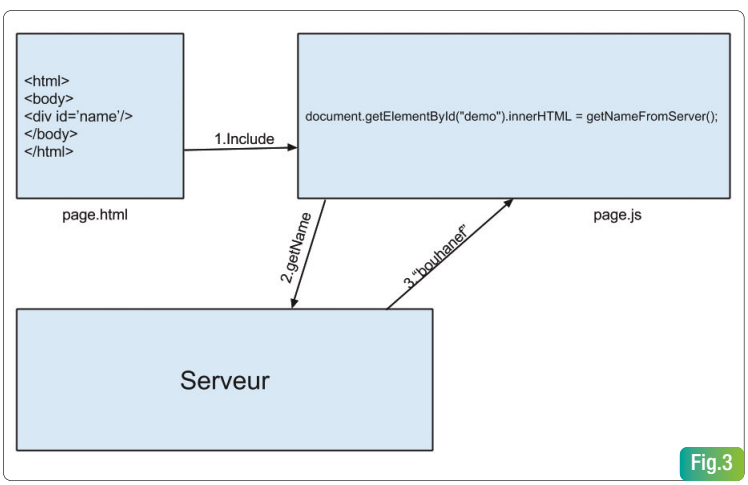


Fig.3

- **A base de colonnes :** Elles ont été créées pour gérer des données réparties sur plusieurs machines, leur principe est de multiplier les clés uniques pour pointer sur plusieurs colonnes. Les colonnes sont disposées par famille. Exemple : Cassandra, HBase, BigTable
- **A base de documents :** Elle est le niveau succédant au modèle clés-valeurs. C'est une collection qui peut contenir des clés-valeurs dont les valeurs peuvent aussi être des collections. Les données sont enregistrées dans des formats JSON. Cela permet une efficacité dans la recherche à l'intérieur des valeurs. Exemple : Lotus Notes, MongoDB
- **A base de graphe :** Pour faciliter l'utilisation des relations (comme les bases de données relationnelles), un système de graphe est mis en place. Cela dépend du fournisseur de base de données; il y a ceux qui proposent une API REST et d'autres qui proposent des APIs de requêtage (intégrant les différents langages de programmation). Mais à ce jour, il n'y a pas de formalisme ou de normalisation pour les requêtes des bases de données NoSQL. Exemple : Allegro

Accès aux données

Plusieurs mécanismes permettent l'accès aux données, cela dépend de l'application, la base de données et le besoin de structuration des données.

- **JDBC (Java DataBase Connectivity) :** C'est un ensemble d'interfaces permettant de normaliser la communication avec les bases de données relationnelles. Il existe des pilotes pour la plupart des fournisseurs de bases de données.
- **JPA (Java Persistance API) :** C'est un ensemble d'interfaces permettant de représenter des bases de données relationnelles sous forme d'objets et classes Java.
- **Hibernate :** C'est un des frameworks les plus utilisés, il permet la persistance des données avec une ou plusieurs bases de données relationnelles.
- **Spring data :** C'est un des modules de Spring, il est constitué de plusieurs sous-modules permettant tous, la gestion des données, NoSQL et relationnelle. Exemples : Spring Data Mongo, Spring Data Hadoop...
- **Objectify :** C'est un framework permettant d'utiliser les bases de données Datastore d'Appengine.

Les couches annexes

Ces couches peuvent être représentées dans les services ou dans une couche à part. Ce sont les modules transverses comme l'impression, la génération des documents (PDF, Excel.), les batchs...

L'impression

Ce module permettra de gérer l'impression des documents, il y a dans java standard l'API Java Print API, cette API est native, elle est facile à mettre en œuvre pour les utilisations basiques mais difficile à configurer.

Le reporting

Toute application générera des documents pour les reporting et la prise de décision...

- **Jasper :** Cet outil permet de créer des templates PDF et de les générer suivant les valeurs données. C'est un outil très puissant qui permet de créer presque tous les types de tableaux et de documents PDF.
- **Birt (Business Intelligence and Reporting Tools) :** C'est un plug-in Eclipse, permettant de créer et de générer des documents de reporting.

La génération des documents (Excel, Word, PDF...)

Un des besoins les plus fréquents est la génération des documents Excel, PDF... L'API la plus utilisée pour ce service est apache POI; il permet de créer et de manipuler des fichiers de la suite bureautique Microsoft Office.

La GED (Gestion électronique des documents)

Pour les rapports générés et plein d'autres documents, on aura besoin de les stocker, les manipuler, les mettre dans l'historique... Pour ce faire, il y a plusieurs outils sur le marché qui permettent de gérer ces besoins, le plus utilisé entre tous est Alfresco, permettant la gestion des documents, il est sous licence libre et assez robuste pour gérer les documents.

Batch

Les batchs sont un enchaînement automatique d'une suite de commandes (processus) sur un ordinateur sans intervention d'un utilisateur... Il y a plusieurs outils permettant de faire des batchs :

- **Spring batch :** C'est un des modules de spring, permettant, avec une simple configuration, de créer et manipuler des batchs.
- **Quartz (scheduler) :** Facile à mettre en place, mais il ne gère que les batchs, tous les autres besoins (Reprise après erreur...) doivent être implémentés explicitement par le développeur.

Les serveurs

Cela dépend des choix technologiques, plusieurs types de serveurs existent :

Docker

Docker, à l'inverse des machines virtuelles, il ne lance pas un système d'exploitation virtuel, mais tourne dans un système d'exploitation et crée un environnement pour déployer une application.

Les serveurs Web

Ce sont des serveurs légers qui permettent d'exécuter du code Java et contenant un serveur http et un moteur de servlet. Les plus connus sur le marché sont Tomcat et Jetty.

Les serveurs d'applications

Ce sont des serveurs qui contiennent des serveurs Web, en plus ils ont un contexte d'exécution de composants (EJB...). Les plus connus sur le marché sont Jboss (ou WildFly), Glassfish, Websphere...

Les serveurs embarqués

On peut avoir des serveurs embarqués, ce sont des serveurs utilisés généralement pour la phase de développement pour ne pas installer un serveur dans la machine; en exécutant un script on peut déployer l'application sur un serveur virtuel créé juste pour les tests. Avec Maven ou Gradle, cela dépend de l'implémentation. Voici un exemple avec Maven dans lequel on peut écrire un script permettant de créer un serveur et déployer l'application automatiquement dans ce serveur. Pour le faire, dans le fichier pom.xml on ajoute le code suivant :

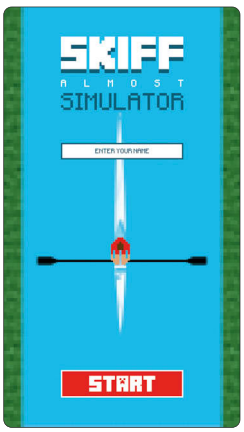
```
...
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.tomcat.maven</groupId>
<artifactId>tomcat7-maven-plugin</artifactId> <version>2.1</version>
      <configuration>
        <path>/</path>
      </configuration>
    </plugin>
  </plugins>
</build>
…
```

Et puis dans la ligne de commande, on exécute :

```
$mvn clean install
$mvn tomcat7:run
```

Connectez votre rameur d'appartement avec Chrome

1ère partie



WiiFit, AppleHealt, Google Fit, tout ça c'est du passé ! Place à SkiffSimulator !! Ça va ramer dans les salons !

Jean-François Garreau

Développeur depuis 10 ans. Responsable du Pole Internet Of Things chez SQLI Nantes depuis 2012, Jean-François est passionné par les nouvelles technologies. Il est également co-fondateur du CDG Nantes, organisateur du DevFest Nantes, co-créateur des Nantes Wit et organisateur de Devoxx4Kids.

Nous allons voir à travers cet article comment réaliser un rameur connecté et ainsi vous permettre de vous amuser en faisant du sport !

La version présentée dans cet article est une V1 dépendant d'un ordinateur. Il pourrait être très facile de faire évoluer l'application pour qu'elle soit autonome sur un équipement de type Raspberry par la suite.

PRINCIPE

Grâce à l'API serial de Google Chrome, nous allons relier directement notre rameur à notre navigateur pour créer un jeu qui nous permettra de jouer à un jeu 8 bits. Voici globalement un schéma symbolisant le montage à effectuer : Nous allons faire communiquer 2 programmes entre eux : Fig.1.

- Un sketch Arduino qui va mesurer la distance du joueur sur le rameur;
- Une Application Chrome avec d'un côté :
  - La partie Chrome App qui va lire le port série;
  - La partie Jeux qui va lire les informations provenant de la Chrome App.

Comme tout ceci n'est pas bien compliqué, j'ai décidé de tout coder from scratch afin de me faire la main sur les possibilités offertes par cet écosystème. Côté application Web, nous avons un simple canvas afin de tirer parti de l'accélération matérielle. Côté matériel, j'ai opté pour une simple Arduino avec un capteur ultrason.

Le matériel

- Un rameur (~20€ sur Le Bon Coin)
- Une Arduinio Nano (~trouvé à 6€ sur tinyDeal)
- Une breadboard (~2€ sur tinyDeal)
- Un capteur ultrason HC-SR04(~1,5€ sur tinyDeal)
- Un fil MiniUSB -> USB (fourni avec l'arduino)
- Des fils pour notre montage
- Un ordinateur avec Chrome

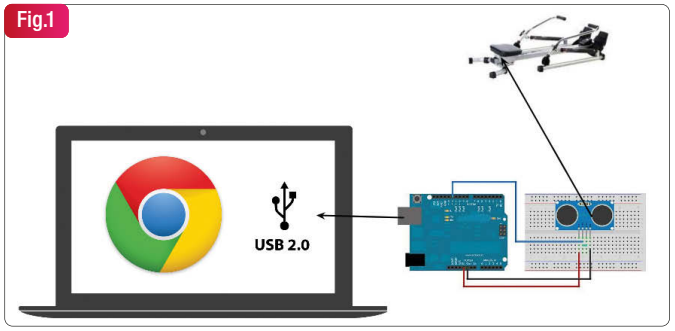
UN JEU EN HTML ?

Avant de commencer, il m'a fallu me renseigner sur le fonctionnement d'un jeu et voir comment j'allais procéder pour respecter au mieux les bonnes pratiques en vigueur. Globalement, un jeu possède plusieurs briques qui fonctionnent en parallèle afin de minimiser le blocage de l'UI. Pour rappel, un jeu est considéré comme fluide s'il est à 60fps ce qui veut dire que chaque affichage ne doit pas dépasser les 13ms. Afin de respecter au mieux cette contrainte, j'ai découpé mon programme :

- La brique qui s'occupe de l'affichage va lire dans un modèle partagé ;
- La brique qui s'occupe de lire les données de l'Arduino va alimenter ce modèle partagé et faire les calculs nécessaires.

De cette façon, j'ai une séparation propre de mes interactions et des actions provenant de l'extérieur pouvant parfois bloquer mon interface. Il est à noter qu'avec ce fonctionnement, je tolère une désynchronisation entre l'état de mon modèle et mon affichage. Je pars du principe que celle-ci sera de maximum 13ms, ce qui est acceptable.

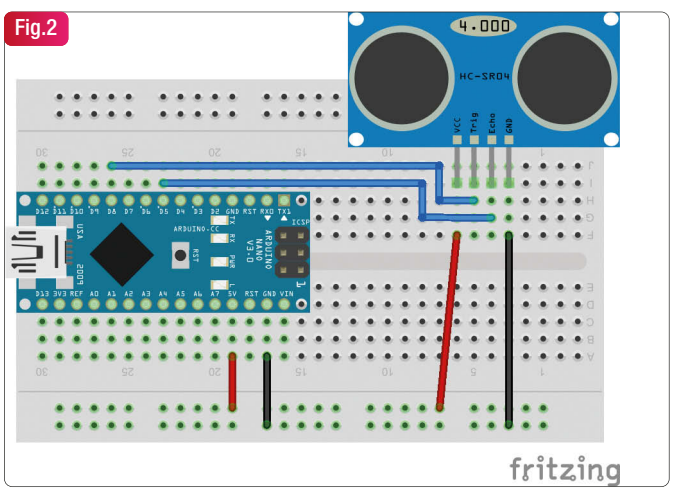
Fig.1



SKETCH ARDUINO Fig.2.

```
//Pour le capteur à ultrasons
int TriggerPin = 8;
//Trig pin
int EchoPin = 5;
//Echo pin
long distance;
void setup() {
  Serial.begin(9600);
  //Mise en entrées de Pins
  //On initialise le capteur à ultrasons
  pinMode(TriggerPin, OUTPUT);
  digitalWrite(TriggerPin, LOW);
  pinMode(EchoPin, INPUT);
  delay(100);
  Serial.println("Fin SETUP capteurs");
  }
void loop() {
  distance = lire_distance();
  Serial.print("D");
  Serial.println(distance);
  //Envoi des données en BT :
```

Fig.2



```
  delay(50);
  }
long lire_distance() {
  long lecture_echo;
  digitalWrite(TriggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(TriggerPin, LOW);
  lecture_echo = pulseIn(EchoPin, HIGH);
  long cm = lecture_echo / 58;
  return(cm);
  }
```

Le fonctionnement est très simple : il suffit de lire la mesure de distance dès que l'on en obtient une, puis on la retranscrit directement sur le port série.

CHROMEAPP ?

Comme il s'agit d'une application Chrome, nous devons créer un fichier manifest.json : Manifest de SkiffSumulator qui correspond au fichier de configuration de l'application Chrome.

Structure de l'application

L'application possède donc plusieurs scripts qui vont tourner en parallèle afin de faire fonctionner le jeu. Voici la structure de mon projet côté application Web :

- assets : répertoire possédant tous les fichiers de ressources du jeu (Fonts, images, sons)
- Javascript : ensemble des scripts Javascript constituant l'application
- scss : fichier sass qui vont servir à générer le css

Nous allons nous attarder uniquement sur les scripts car c'est dans cette partie que se situe toute l'intelligence du jeu. En effet, le fichier html est très sommaire car il ne contient qu'un canvas :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Skiff Almost Simulator</title>
    <link rel="icon" type="image/png" href="./assets/images/icon48.png" />
    <link rel="stylesheet"href="./css/app.css"/>
  </head>
  <body>
    <canvas id="skiff"></canvas>
    <input id="user" type="text" placeholder='Enter your name'/>
    <script type="text/javascript" src="./javascript/resources.js"></script>
    <script type="text/javascript" src="./javascript/audio.js"></script>
    <script type="text/javascript" src="./javascript/const.js"></script>
    <script type="text/javascript" src="./javascript/chrome_storage.js"></script>
    <script type="text/javascript" src="./javascript/chrome_serial.js"></script>
    <script type="text/javascript" src="./javascript/screen_accueil.js"></script>
    <script type="text/javascript" src="./javascript/screen_action.js"></script>
    <script type="text/javascript" src="./javascript/screen_end.js"></script>
    <script type="text/javascript" src="./javascript/app.js"></script>
  </body>
</html>
```

Scripts et rôles

Voici les différents fichiers et leurs rôles :

- app.js : coeur de l'application, il s'agit du point d'entrée de l'application et il agit comme un chef d'orchestre. C'est dans un sens le contrôleur de notre application;
- audio.js : fichier servant à gérer la lecture des fichiers audio;
- chrome_serial.js : fichier contenant le code spécifique à Chrome qui va nous permettre de lire directement depuis le port série de l'ordinateur;
- chrome_storage.js : fichier utilitaire qui expose de façon uniforme une API de localstorage au cas où l'application devrait tourner en dehors de Chrome (plus de détails plus loin dans l'article);
- const.js : fichier regroupant toutes les constantes du jeu. Il peut s'agir de simples constantes ou de variables d'ajustement servant lors de la calibration du jeu;
- ressources.js : fichier permettant d'exposer un mécanisme de chargement de ressources graphiques en vue de les exploiter par la suite dans le programme;
- screen_acceuil.js : fichier contenant tout le code spécifique à l'affichage de l'écran d'accueil;
- screen_action.js : fichier contenant tout le code spécifique à l'affichage pendant le jeu;
- screen_end.js : fichier contenant tout le code spécifique à l'affichage de l'écran de fin.

Le Reveal Module Pattern a été choisi comme pattern car il permet de fonctionner en module Javascript et d'offrir un découpage propre du code tout en maîtrisant les méthodes exposées.

DÉROULEMENT DU PROGRAMME :

Prenons les différents points méritant de l'attention :

Démarage (App.js)

```
//API
function init() {
  window.addEventListener('load',
  pageLoad);
  }
return {
  init : init, ... }
}
();
AppSAS.init();
```

On démarre l'application dès que la page est prête.

```
'use strict';
var AppSAS = AppSAS || function() {
  ... function pageLoad() {
    // On se connecte à l'Arduino
    try {
      skiffSimulatorChrome.initArduino();
      }
    catch(err) {
      console.error("Error : %s \n %s",
      err.message, err.stack);
      }
```

On doit faire appel au module qui va lire les données de l'Arduino.

```
// On initialise le canvas
ui.input = document.getElementById('user');
ui.canvas = document.getElementById('skiff');
ui.canvas.width = window.innerWidth;
ui.canvas.height = window.innerHeight;
```

```
ui.context = ui.canvas.getContext('2d');
ui.canvas.addEventListener('click', checkClick, false);
// On précharge toutes les ressources nécessaires
ui.resources.loadSprites([ {
  title : 'logo', url : 'assets/images/logo.png'}
, {
  title : 'game_over', url : 'assets/images/gameover.png'}
, {
  title : 'rive_gauche_portrait', url : 'assets/images/riviere_gauche_portrait.png'}
, ... ]) .then(function(value) {
  paintSkiff();
  }
).catch(function(err) {
  console.error("Error : %s \n %s", err.message, err.stack); }
);
}
```

On initialise notre canvas ainsi que les ressources graphiques du projet. On n'affiche le jeu qu'une fois ces dernières chargées.

En fonctionnement (app.js)

Une fois l'application réellement démarrée avec la méthode paintSkiff. Nous allons simplement déléguer l'affichage aux méthodes appropriées :

```
// Gère l'affichage de l'écran
function paintSkiff() {
  try {
    ... // Affichage des décors
    paintBackground();
    if (gameModel.stateGame === constState.STATE_ACCUEIL) {
      ScreenSasAccueil.paintSkiffAccueil();
      StorageSAS.manageGhost();
      }
    else if (gameModel.stateGame === constState.STATE_RUNNING) {
      // On doit peindre le fantôme du jeu en deuxième car son alpha nous indique où il est
      ScreenSasAction.paintSkiffAction();
      ScreenSasAction.paintSkiffGhost();
      // On ajoute l'état à l'historique
      gameModel.currentHistory.push( {
        direction : gameModel.direction,
        distanceSkiff : + gameModel.distanceSkiff,
        distanceArduino : + gameModel.distanceArduino }
      );
      gameModel.step++;
      }
    if (gameModel.stateGame === constState.STATE_END) {
      ScreenSasEnd.paintSkiffEnd();
      }
    window.requestAnimationFrame(paintSkiff);
    }
  catch (err) {
    console.error("Error : %s \n %s",
    err.message, err.stack);
    }
  }
```

Il est à noter que l'on utilise la méthode window.requestAnimtionFrame. Cette dernière est très importante car elle permet d'optimiser l'affichage de nos écrans en fonction de la puissance de la machine. En effet, la méthode de callback ne sera appelée qu'une fois le navigateur prêt à effectuer une nouvelle mise à jour graphique. Il faut donc utiliser cette méthode à la place d'un setInterval.

Affichage des écrans

L'affichage des écrans se fait toujours de la même façon :

- On nettoie le canvas
- On dessine une ou des images sur le canvas

L'affichage d'un écran se fait toujours de la façon suivante :

```
ui.context.drawImage(imgSource //L'image source
, sx //sx clipping de l'image originale
, sy //sy clipping de l'image originale
, sw // swidth clipping de l'image originale
, sh // sheight clipping de l'image originale
, dx // x Coordonnées dans le dessin du canvas
, dy // y Coordonnées dans le dessing du canvas
, dw // width taille du dessin
, dh // height taille du dessin
);
```

Les animations / constructions des écrans ne sont en fait qu'une succession de drawImage ou fillText Fig.4.

Gestion des interactions

Afin de pouvoir démarrer l'application, nous devons gérer les clics sur le canvas. Le problème est que lorsque nous dessinons des images, nous ne pouvons pas avoir accès à un équivalent de onClick sur une zone graphique précise. Nous devons donc écouter les clicks sur le canvas et calculer si la zone de clic correspond à une zone d'interaction de notre ihm.

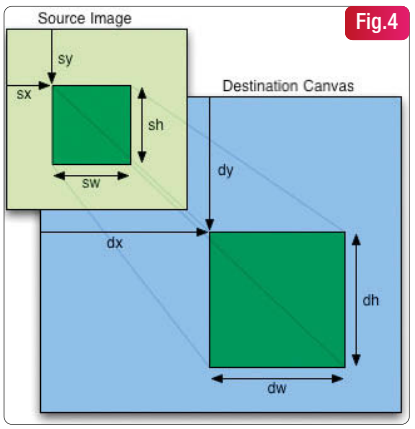


Fig.4

```
                // Gère les clics en fonction de l'état du jeu
function checkClick(event) {
  if (gameModel.stateGame != constState.STATE_RUNNING) {
    var btnStart = ui.resources.images['btn_start'];
    var finalHeight = btnStart.height * ui.ratio,
    finalWidth = btnStart.width * ui.ratio;
    var x = (ui.canvas.width / 2) - ((btnStart.width * ui.ratio) / 2), y = ui.canvas.height - finalHeight
- (isPortrait() ? 100 : 50);
    var xClick = event.pageX,
    yClick = event.pageY;
    if (yClick > y && yClick < (y + finalHeight) && xClick > x && xClick < (x + finalWidth)) {
      // On change l'état du jeu
      gameModel.stateGame = gameModel.stateGame === constState.STATE_ACCUEIL ? constState.
STATE_RUNNING : constState.STATE_ACCUEIL;
      ...
      }
    }
  }
```

Suite et fin du projet rameur le mois prochain.

Prenez vos repères avec Google Maps API V3

Initialement lancé aux USA en 2004, Google Maps débarque en France le 27 Avril 2006 en version bêta, il est considéré comme stable depuis septembre 2007.

Vincent Quadrelli,
Osaxis

Google Maps fait partie de la suite logicielle de Google qui propose des services tels que le Street View, permettant d'observer un panorama à 360° des zones urbaines et rurales. Il y a aussi Google Moon et Mars qui sont respectivement des visualisations de la lune et de la planète Mars. Enfin, Google Earth qui offre une visualisation de la planète Terre par le biais de vues satellitaires et aériennes.
Nous nous attarderons uniquement sur le service Google Maps et plus particulièrement sur l'implémentation de ce service sur notre site Web, grâce à l'API JavaScript V3 fournie par Google dont la première version V3.19 est sortie en février 2015.

Notre projet pratique

En dehors de son API et de ses nombreux services, Google, comme toute multinationale de ce nom, est un généreux mécène qui multiplie les partenariats avec les musées. Je vous propose donc à travers ce projet de créer une Google Map référençant les divers musées parisiens.

Hello Map

Nous nous retrouvons dans notre IDE favori avec une page HTML vierge que nous allons un peu égayer en ajoutant la description du projet :

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>La chasse aux musées</title>
  </head>
  <body>
    <div id="description">
      <h1>La chasse aux musées</h1>
      <span>avec</span>
      <h3>Google Maps JavaScript API</h3>
      <p>Le but de cette page est de recenser les différents musées Parisiens !</p>
    </div>
  </body>
</html>
```

Alors, je vous ai promis une carte et elle arrive, mais avant cela une petite explication du code nécessaire à son affichage :

- Une div pour accueillir notre carte qui portera l'id #map-canvas (évidemment vous êtes libre de le changer, mais il faudra répercuter cela tout au long du projet) :

```
<div id="map-canvas"></div>
```

- Le CDN (Content Delivery Network) de Google Maps API V3 qui nous permet d'utiliser en une ligne l'API :

```
<!-- CDN Google Map Api -->
<script src="https://maps.googleapis.com/maps/api/js?v=3.exp"></script>
```

- Et évidemment un petit bout de JavaScript que nous détaillerons par la suite,
- Nous ajouterons un soupçon de CSS tout de même pour la forme :

```
<style type="text/css">
    html, body, #map-canvas {
      height: 100%;
      margin: 0;
      padding: 0;
      font-family: cursive;
    }
    #description {
      text-align: center;
      position: absolute;
      font-size: 13px;
      z-index: 9;
      right: 20px;
      top: 20px;
      background-color: rgba(166, 166, 166, 0.88);
      border-radius: 20px;
      color: rgb(58, 57, 61);
    }
  </style>
```

L'API nous propose d'ajouter des options, nous utiliserons les options « center » et « zoom » qui nous permettent respectivement de choisir le centre de la carte à l'initialisation, et le zoom souhaité sur ce point.

```
<script type="text/javascript">
  /* Déclaration de la Map */
  var map;
  /* Déclaration des options et initialisation de la carte */
  function initialize() {
     var mapOptions =
     {
     zoom: 13,
     center: new google.maps.LatLng(48.859952,2.335068)
      };
```

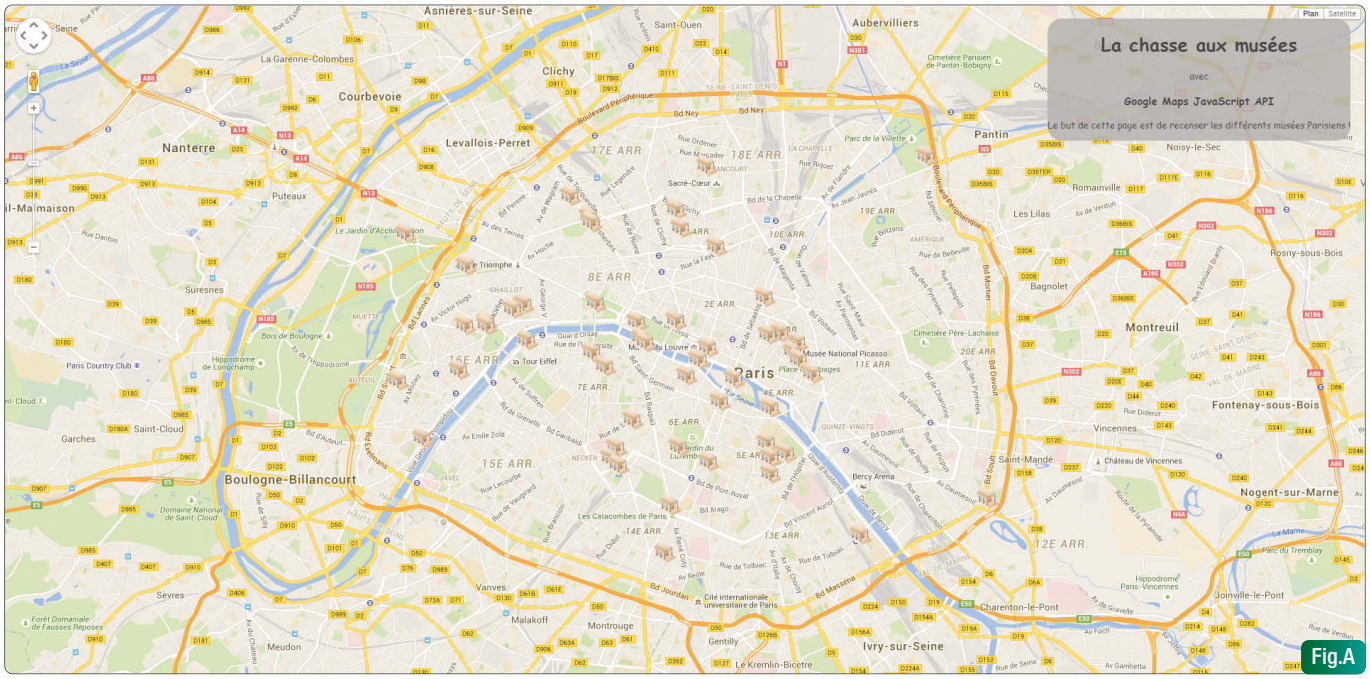
La carte est initialisée en passant le DOM #map-canvas à l'API Google Map.

```
map = new google.maps.Map(
      document.getElementById('map-canvas'),mapOptions);
    }
```

Parfait : nous avons donc une fonction qui nous permet d'afficher notre super carte ! Il ne nous reste plus qu'à appeler cette fonction. Nous ajoutons donc un « Listener » afin d'initialiser la carte au chargement de la page.

```
/* Listener de la fonction load pour lancer l'initialisation de la Map */
  google.maps.event.addDomListener(window,'load', initialize);
</script>
```

Nous voilà donc avec une carte certes très jolie, mais où sont donc nos musées ? On va essayer d'éviter le placement et le référencement à la main... Pour cela nous utiliserons la source libre opendata.paris.fr qui nous propose de nombreux sets de données de tous genres pour la ville de Paris. Vous pouvez télécharger le set utilisé dans cet exemple à cette URL : http://opendata.paris.fr/explore/dataset/liste-musees-de-france-a-paris/?tab=export



Carte de musées parisiens au 5 juin 2015

Du JSON (JavaScript Object Notation) au marqueur

Voilà, nous avons notre fichier Json que nous intégrons à notre projet sous le nom de 'markers.json'.

Et nous allons ajouter la librairie jQuery afin de parcourir simplement notre JSON.

```
<!--CDN Jquery-->
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
```

Nous y voilà, pour déclarer un marqueur et donc repérer nos musées, nous n'avons besoin que d'une paire de coordonnées et d'une carte sur lesquels les afficher. Il nous suffit donc de parcourir le fichier Json par l'intermédiaire de la fonction jQuery « $.getJSON ». Nous vérifions la présence de coordonnées au cas où le fichier Json serait erroné (notre fichier en l'occurrence pour le musée n°39 lors de l'écriture de cet article).

```
<script type="text/javascript">
/* Récupération du Json, lecture du Json, vérification de la présence de coordonnées et
 ajout de markers */
  $.getJSON('markers.json', function (data) {
    for (var i in data) {
      if (data[i].fields.hasOwnProperty("coordonnees_")) {
```

Puis à chaque itération, nous instancions un marqueur et lui passons en paramètre une position, formée grâce aux champs coordonnées_ 0 et 1 de l'objet Field de notre fichier JSON et de notre carte précédemment instanciée.

```
var myLatLng = new google.maps.LatLng(
  data[i].fields.coordonnees_[0],
  data[i].fields.coordonnees_[1]);  /*creation de la position*/
var marker = new google.maps.Marker({
            position: myLatLng,
            map: map,
            icon: 'museum_32.png'
        }); /*création du marqueur*/
      } /* vérification de la présence de coordonnées */
    }  /* boucle sur les coordonnées des marqueurs */
  }); /* lecture du fichier Json */
  /* listener de la fonction load pour lancer l'initialisation de la Map */
  google.maps.event.addDomListener(window, 'load', initial-ize);
</script>
```

Nous y sommes, notre carte des musées parisiens est prête, voyons le résultat. Dans cet exemple nous utiliserons les valeurs suivantes.

zoom = 13;

center = 48.859952 , 2.335068 (Paris)

image = 'museum_32.png' (image ajoutée au projet) **Fig.A.**

Conclusion et approfondissement

Google nous offre, par le biais de son API Google Maps, la possibilité de mettre en place simplement et rapidement une carte personnalisable tant esthétiquement que fonctionnellement.

Pour tous types de données de services et d'utilisations, il nous sera possible de créer une carte sur mesure. De même, le Json se marie très bien avec cet outil et apporte performance et compatibilité à vos applications. A noter qu'il existe un dérivé du Json, le GeoJson, spécification du Json orientée pour les données géographiques.

Pour aller plus loin, il est même possible de stocker en base de données les coordonnées, de générer le Json lors d'un appel Ajax, et garder ce principe pour l'affichage de fenêtres contenant les informations du marqueur. Il est envisageable de stocker dans une autre table les informations liées à chaque marqueur afin de les afficher lors d'un clic sur le marqueur donné. Pour cela, Google met à disposition les Info-Windows (pour plus d'informations, la documentation de Google se trouve à l'adresse suivante : https://developers.google.com/maps/documentation/JavaScript/markers).

En résumé, les possibilités sont quasiment illimitées, contrairement à cet article qui touche à sa fin. Bon dev à tous avec Google Maps !

Node.js de A à Z

3e partie

Je ne vous apprends rien en vous disant que beaucoup de développeurs Web utilisent JavaScript pour créer des applications front. Node.js permet à ce langage très populaire d'être utilisé dans plusieurs autres contextes, par exemple sur un serveur Web. Il existe plusieurs fonctionnalités notables offertes par Node.js ce qui le rend sans nul doute digne d'un grand intérêt.

Wassim Chegham (@manekinekko)
Expert en Nouvelles Technologies Web chez Groupe SII
Google Developer Expert (GDE) en AngularJS

DÉBOGUER ET TESTER

Je me rappelle d'un de mes collègues qui me répétait souvent que "tester c'est douter !". Personnellement, je préfère douter maintenant tant qu'il est encore temps que de le regretter plus tard, lorsqu'un bout de l'application tombe en pleine Prod.

Dans l'écosystème Node, il existe une multitude de modules nous permettant de tester notre application à différents niveaux. Ces modules respectent la philosophie Node qui stipule qu'un module doit être :

- Petit,
- Interchangeable,
- Spécialisé dans une seule tâche.

Ce kit d'outils est donc une collection de petits modules Node organisée comme suit :

- Frameworks de Tests : Intern, Vows, Mocha.
- Librairies d'assertions : Chai, Assert.
- Stubs : Sinon.

Fig.65

```
programmez.spec.js — programmez
programmez.js | programmez.spec.js
1  describe('Programmez', () => {
2
3    before(() => {});
4
5    beforeEach(() => {});
6
7    it('devrait afficher "Programmez!"', () => {
8      /* Votre test ici */
9    });
10
11   after(() => {});
12
13 });
Line 1, Column 1   0 misspelled words   Tab Size: 2   JavaScript
```

Fig.66

```
programmez.spec.js — programmez
programmez.js | programmez.spec.js
1  var assert = require('assert');
2  var chai = require('chai');
3
4  describe('Programmez', () => {
5    var foo = '';
6    before(() => foo = 'Programmez' );
7
8    it('foo devrait valoir "Programmez!" (Assert)', () => {
9      assert(foo === 'Programmez', 'OK');
10   });
11   it('foo devrait valoir "Programmez!" (BDD)', () => {
12     chai.expect(foo).to.equal('Programmez');
13   });
14   it('foo devrait valoir "Programmez!" (TDD)', () => {
15     chai.assert.equal(foo, 'Programmez');
16   });
17
18 });
Line 1, Column 1   3 misspelled words   Tab Size: 2   JavaScript
```

Nous n'allons pas détailler le fonctionnement de chaque module. Mais, nous allons les présenter brièvement et parler de la philosophie de chacun.

Frameworks de Tests

Avant toute chose, avant même de commencer à écrire vos tests, il va vous falloir un framework de Tests. Ce framework va vous apporter toutes les fondations nécessaires pour exécuter et orchestrer vos tests. Sans surprise, il existe énormément de frameworks, mais Mocha est le plus connu, il est là depuis les débuts de Node. Ce framework est très flexible et très configurable. Voici comment se présente un test avec Mocha : Fig 65

Explications :

- Le bloc **before()** est appelé avant le début des tests dans la campagne de test **"Programmez"**. Ce bloc permet d'initialiser la configuration de la campagne.
- Le bloc **beforeEach()** est appelé avant chaque test de cette campagne.
- Les blocs **it()** contiennent votre test
- Le bloc **after()** est appelé après la fin de tous les tests. Ce bloc permet de faire le ménage.

Librairies d'assertions

Une fois que le framework est en place, vous êtes prêt(e)s à écrire vos tests. Le plus simple est d'utiliser une librairie d'assertion. Node propose un module **Assert** de base permettant d'écrire des tests à base d'assertions. Il existe cependant plusieurs syntaxes possibles : TDD, BDD, assert(), should()...etc.

Ce sera à vous de choisir la syntaxe (et donc le module) qui vous convient, mais sachez que BDD a le vent en poupe grâce à sa syntaxe ressemblant au langage naturel. Le module Chai est un excellent module qui supporte la plupart de ces syntaxes. Fig 66 et 67

Stubs

Faire de simples assertions n'est malheureusement pas suffisant pour tester des fonctions un peu trop complexes. Il peut arriver que l'on veuille influencer l'exécution d'une fonction pour tester des cas limites ou juste tordus. Le module Sinon nous permet de réaliser cela. Fig 68 et 69

Sinon permet de faire beaucoup plus que cela. Il offre par exemple des **Spies** qui permettent de surveiller quand une fonction a été invoquée et avec quels arguments. Il offre aussi des **Mocks** pour les bouchons, des timers..etc. Je vous laisse essayer et expérimenter ce module. Vous risquez

Fig.67

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ ./run.spec.sh

Programmez
  ✓ foo devrait valoir "Programmez!" (Assert)
  ✓ foo devrait valoir "Programmez!" (BDD)
  ✓ foo devrait valoir "Programmez!" (TDD)

3 passing (10ms)

wchegham 192.168.0.27  /tmp/programmez
$
```

certainement de l'adopter. Ces modules font partie d'un petit échantillon de ce que nous offre la communauté Node en matière d'outils de test. Si vous souhaitez avoir un seul outil qui regroupe tous ces modules voire bien plus, je vous recommande **Intern.io**. Une image vaut mieux qu'un long discours : Fig 70. Mon collègue Vincent a écrit un article expliquant comment démarrer avec Intern.io, il est accessible à cette adresse [15].

DÉPLOYER EN PRODUCTION

Maintenant que votre application est prête, il est temps de la mettre en production et de la mettre à disposition de millions d'utilisateurs.

Voici le diagramme de l'architecture cible que nous allons découvrir ensemble dans cette section : Fig 71

Notre application Node tourne sur un serveur dans un réseau local (privé). Cette application écoute le port 3000 et n'est pas accessible directement par le Web. Nous utilisons un serveur Web en tant que *Reverse Proxy*. Ce serveur lui écoute sur le port 80.

Dans la suite de cette section, nous allons parcourir ensemble les différents points à vérifier pour mettre en production une application Node. Voyez cette section comme une sorte de *Checklist* de mise en production.

Fig.68

```
programmez.spec.js — programmez
programmez.js   programmez.spec.js
1  var sinon = require('sinon');
2  var chai = require('chai');
3
4  describe('Programmez', () => {
5
6    it('foo devrait valoir "Programmez!" (Sinon)', () => {
7
8      var foo = sinon.stub().returns('Programmez');
9      var proxy = foo();
10
11     chai.expect(proxy).to.equal('Programmez');
12
13   });
14
15 });
Line 1, Column 1   1 misspelled word   Tab Size: 2   JavaScript
```

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ ./run.spec.sh

  Programmez
    ✓ foo devrait valoir "Programmez!" (Sinon)

  1 passing (11ms)

wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.69

Feature	Intern	QUnit	Mocha	Jasmine	BusterJS	Karma	Nightwatch.js
Includes unit testing	Yes	Yes	Yes	Yes	Yes	No	No
Includes functional testing	Yes	No	No	No	No	No	Yes
Code coverage analysis	Yes	No	Yes	No	Extension	Yes	No
True[1] browser events	Yes	No	No	No	No	No	Yes
Native AMD support	Yes	No	No	No	Extension	Extension	No
Stand-alone[2] browser support	Yes	Yes	Build required	Build required	Experimental	No	No
Node.js support	Yes	No[3]	Yes	Yes	Yes	Yes	No
Source map support	Yes	No	No	No	No	No	No
Any[4] assertion library	Yes	No	Yes	No	Yes	N/A	No
Default test interface	TDD, BDD, object	TDD	TDD, BDD, object	BDD	TDD, BDD	N/A	Object
Extensible test interfaces	Yes	No	Yes	No	Yes	N/A	Commands only
Extensible reporters	Yes	No	Yes	No	Yes	N/A	No
Asynchronous support	Promises	Globals	Callbacks, Promises	Polling	Callbacks, Promises	Callbacks	Callbacks
Selenium support	Yes	No	No	No	No	No	Incomplete
Fixes Selenium bugs	Yes	No	No	No	No	No	No
Tests native mobile apps	Yes	No	No	No	No	No	Yes
Built-in CI support	Yes	No	No	No	Yes	Yes	Partial
Built-in Sauce Labs integration	Yes	No	No	No	No	No	No
Built-in BrowserStack integration	Yes	No	No	No	No	No	No
Built-in TestingBot integration	Yes	No	No	No	No	No	No
Built-in Travis CI integration	Yes	No	No	No	No	Yes	No
Grunt support	Yes	3rd party					

Fig.70

Configuration du serveur

Évidemment je ne devrais pas revenir sur cette partie, vous aurez besoin de Node et de NPM pour faire tourner votre application en production.

Vous pouvez utiliser NVM pour gérer les différentes versions de Node (voir la première partie de ce dossier). Vous aurez besoin de NPM pour récupérer les dépendances déclarées dans le fichier **package.json**. Une chose très importante : pensez à bien figer les versions des dépendances. Pour cela, je vous recommande d'utiliser le module **shrinkwrap** [16] prévu à cet effet. Passons maintenant à la gestion du démarrage et d'arrêt de votre (vos) application(s) Node.

Installation de PM2

PM2 [17] est un gestionnaire de processus pour les applications Node. PM2 offre une interface très simple pour gérer les applications facilement, mais aussi en tant que services.

Une fois installé, PM2 vous permet de lancer votre application comme ceci : Fig 72

PM2 affecte directement un PID, il affiche aussi d'autres informations utiles telles que la consommation de la mémoire, le statut de l'application, etc. L'intérêt de PM2 est que lorsque l'application Node crashe, elle est automatiquement redémarrée. PM2 offre également la possibilité de gérer l'application au démarrage du système. Lorsque votre serveur reboote par exemple, PM2 est capable de lancer automatiquement votre application ce qui peut être très utile. Pour cela nous utilisons la sous-commande **startup** de PM2 : Fig 73

Il existe d'autres sous-commandes proposées par PM2, je vous laisse les découvrir à tête reposée.

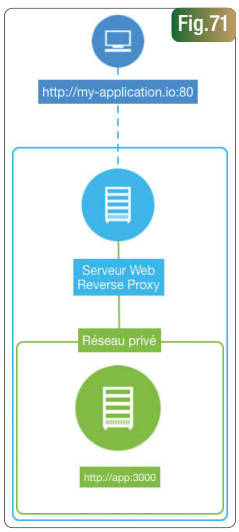


Fig.71

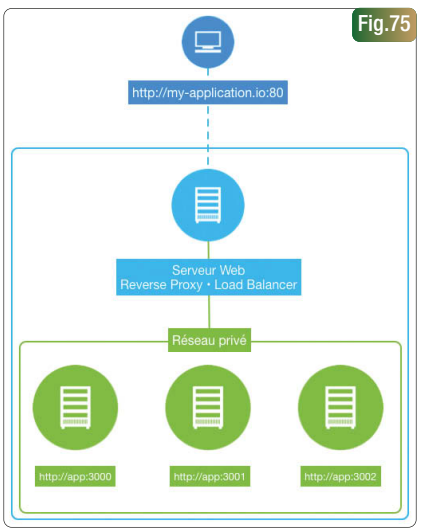


Fig.75

```
wchegham — /Users/wchegham — | Users/programmez — bash — 93×14
wchegham at wchegham-mbp-2.local 192.168.0.27 /tmp/programmez
$ pm2 start app.js --name Programmez
[PM2] Starting app.js in fork_mode (1 instance)
[PM2] Done.
```

App name	id	mode	pid	status	restart	uptime	memory	watching
[redacted]	1	fork	25373	online	0	3m	36.707 MB	disabled
[redacted]	2	fork	25456	online	0	3m	80.723 MB	disabled
[redacted]	3	fork	0	stopped	0	0	0 B	disabled
Programmez	4	fork	27255	online	0	0s	8.695 MB	disabled

```
Use `pm2 show <id|name>` to get more details about an app
```

Fig.72

```
wchegham — /Users/wchegham — | Users/programmez — bash — 91×7
wchegham at wchegham-mbp-2.local 192.168.0.27 /tmp/programmez
$ pm2 startup
[PM2] You have to run this command as root. Execute the following command:
      sudo env PATH=$PATH:/usr/local/bin pm2 startup darwin -u wchegham

wchegham at wchegham-mbp-2.local 192.168.0.27 /tmp/programmez
$
```

Fig.73

Configuration du Reverse Proxy

Maintenant que votre application est démarrée et qu'elle écoute sur un port du réseau privé. Nous allons ajouter un serveur Web qui va la rendre accessible depuis le Web. Pour cela, nous allons utiliser **nginx** en tant que *Reverse Proxy*. Fig 74

Grâce à cette configuration (très minimale), nous allons pouvoir accéder à notre application via l'adresse http://my-application.com. Le serveur Web va ainsi *proxifier* les requêtes vers l'application Node, définies dans le bloc **upstream**.

Félicitations ! Votre application Node tourne en production !

Mise à échelle de l'application

Configuration de nginx

Grâce à cet exemple d'architecture, il sera possible de mettre simplement (tout est relatif) votre application à l'échelle. Il se trouve que nginx intègre de base un Load Balancer. Mais sachez qu'il existe d'autres solutions que vous pouvez utiliser à la place de nginx : HAProxy [18], Varnish [19] (en plus de sa fonction première qui est serveur de cache) et même PM2 possède une fonctionnalité de Load Balancing entre processus.

Voici le diagramme précédent revu avec cette nouvelle contrainte : Fig 75

La différence par rapport au premier diagramme est que nous avons ajouté d'autres instances de notre application, chacune écoute sur un port différent, 3000, 3001 et 3002, etc. Nous allons ensuite mettre à jour le fichier de configuration de nginx : Fig 76

La directive **ip_hash** nous permet de nous assurer que les requêtes provenant du même client seront passées au même serveur (sauf s'il est indisponible). Pour réaliser cela, nginx utilise l'IP du client comme clé de hash.

Aller plus loin avec PM2

En plus de démarrer plusieurs instances de notre application, dont chacune va écouter un port différent, nous pouvons utiliser PM2 pour lancer notre application en mode **cluster** afin de profiter de tous les coeurs du CPU de notre machine.

Avec PM2, nous allons lancer notre application avec un paramètre supplémentaire.

Grâce à ce mode, PM2 vient de lancer autant d'instances de mon application que de coeurs disponibles dans le CPU de mon Macbook Pro, c'est-à-dire huit. Mais je peux dire à PM2 d'augmenter ce nombre, avec la sous-commande **scale** : Fig 77

Votre application tourne en production et en nombre !

Fig.74

```
upstream programmez_node_de_a_z {
    server programmez.local:3000;
}
server {
    location / {
        proxy_set_header        Upgrade $http_upgrade;
        proxy_set_header        Connection "upgrade";
        proxy_http_version      1.1;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header        Host $host;
        proxy_pass              http://programmez_node_de_a_z;
    }
}
```

Fig.76

```
upstream programmez_node_de_a_z {
    ip_hash;
    server programmez.local:3000;
    server programmez.local:3001;
    server programmez.local:3002;
}
server {
    location / {
        proxy_set_header        Upgrade $http_upgrade;
        proxy_set_header        Connection "upgrade";
        proxy_http_version      1.1;
        proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header        Host $host;
        proxy_pass              http://programmez_node_de_a_z;
    }
}
```

CONCLUSION

Nous voilà déjà arrivés à la fin de cette série d'articles consacrée essentiellement à la découverte de Node.js en mode *vanilla* — c'est-a-dire sans framework. Bien évidemment, dans la pratique, et pour des applications métiers relativement complexes, je vous recommande d'utiliser un framework ou des librairies pour vous faciliter le travail. Le but du jeu est de profiter de tous les avantages de Node sans souffrance. Dites-vous toujours que lorsque vous avez un problème à résoudre, il existe sûrement un module pour vous aider. D'ailleurs voici en bonus, une liste à jour des modules Node les plus populaires et les plus intéressants [21].

À vos Nodes, prêts, codez...

Remerciements

Je tiens à remercier tout particulièrement mes deux collègues Benoit Colombani et Jaffar Boudad pour leur patience et leurs relectures à la fois orthographique et technique.

Icônes

Les icônes utilisées dans les schémas proviennent du site : http://flaticons.net/

Fig.77

```
wchegham at wchegham-mbp-2.local 192.168.0.27 /tmp/programmez
$ pm2 scale Programmez 20
[PM2] Scaling up application
[PM2] Scaling up application
[PM2] Scaling up application
[PM2] Scaling up application
[PM2] Scaling up application
[PM2] Scaling up application
[PM2] Scaling up application
[PM2] Scaling up application
[PM2] Scaling up application
[PM2] Scaling up application
[PM2] Scaling up application
[PM2] Scaling up application
```

App name	id	mode	pid	status	restart	uptime	memory	watching
[illegible]	1	fork	25373	online	0	17m	39.504 MB	disabled
[illegible]	2	fork	25456	online	0	16m	84.793 MB	disabled
[illegible]	3	fork	0	stopped	0	0	0 B	disabled
Programmez	115	cluster	28574	online	0	56s	25.609 MB	disabled
Programmez	116	cluster	28575	online	0	56s	24.918 MB	disabled
Programmez	117	cluster	28576	online	0	56s	24.813 MB	disabled
Programmez	118	cluster	28577	online	0	56s	24.941 MB	disabled
Programmez	119	cluster	28578	online	0	56s	25.270 MB	disabled
Programmez	120	cluster	28579	online	0	56s	25.637 MB	disabled
Programmez	121	cluster	28580	online	0	56s	25.473 MB	disabled
Programmez	122	cluster	28581	online	0	56s	25.406 MB	disabled
Programmez	123	cluster	28636	online	0	1s	24.746 MB	disabled
Programmez	124	cluster	28637	online	0	1s	24.730 MB	disabled
Programmez	125	cluster	28638	online	0	1s	24.906 MB	disabled
Programmez	126	cluster	28639	online	0	1s	24.898 MB	disabled
Programmez	127	cluster	28640	online	0	1s	24.762 MB	disabled
Programmez	128	cluster	28641	online	0	0s	24.730 MB	disabled
Programmez	129	cluster	28642	online	0	0s	24.449 MB	disabled
Programmez	130	cluster	28643	online	0	0s	24.555 MB	disabled
Programmez	131	cluster	28644	online	0	0s	24.293 MB	disabled
Programmez	132	cluster	28645	online	0	0s	24.324 MB	disabled
Programmez	133	cluster	28646	online	0	0s	24.223 MB	disabled
Programmez	134	cluster	28647	online	0	0s	23.766 MB	disabled

```
Use `pm2 show <id|name>` to get more details about an app
wchegham at wchegham-mbp-2.local 192.168.0.27 /tmp/programmez
$
```

Références

[15] https://blog.groupe-sii.com/the-intern-retour-sur-la-prochaine-generation-doutils-de-tests-javascript/

[16] https://github.com/joyent/node/wiki/projects,-applications,-and-companies-using-node

[17] https://docs.npmjs.com/cli/shrinkwrap

[18] https://github.com/Unitech/pm2

[19] http://www.haproxy.org/

[20] https://www.varnish-cache.org/

[21] https://github.com/sindresorhus/awesome-nodejs

Les outils Webmaker de Mozilla

En toute saison, il ne faut pas perdre la main avec le Web et surtout ne pas s'éloigner de celui-ci. Nous vous proposons de découvrir quelques outils libres de la fondation Mozilla, regroupés sur un portail Web appelé 'webmaker'.

Christophe Villeneuve
Consultant IT pour Neuros, auteur du livre "Drupal avancé" aux éditions Eyrolles et auteur aux Editions ENI, Rédacteur pour WebRIVER, membre des Teams DrupalFR, AFUP, LeMug.fr (MySQL/MariaDB User Group FR), Drupagora, PHPTV.

LES OUTILS

Popcorn Maker

Popcorn Maker va vous permettre de remixer des vidéos, des sons et des images à travers votre navigateur Firefox et se présente sous la forme d'un éditeur vidéo. Ainsi, vous pouvez intégrer différents types de médias externes Youtube, Vimeo, SoundCloud, d'un média HTML5, des images ou des photos, etc.
Pour utiliser Popcorn Maker, vous restez dans le navigateur car tout est géré avec votre souris, et le glisser-déposer. Vous pouvez sélectionner et ajouter vos propres commentaires, messages et liens dans votre vidéo. L'application Popcorn Maker se présente en 3 parties Fig.1.
Tout d'abord, la partie centrale s'occupe d'afficher le rendu de ce que vous avez construit. Cette zone permet de déterminer les dimensions de chaque objet que vous aurez défini. La partie basse de l'écran est une gestion de calques, c'est à dire que vous pouvez obtenir différentes lignes pour afficher plusieurs contenus en même temps comme l'affichage d'une image, d'un texte et d'une musique ou d'une vidéo Fig.2.
La partie de droite propose les ressources disponibles. Tout d'abord, vous pouvez enregistrer les liens vidéo dans la bibliothèque pour être utilisés ultérieurement. Ensuite, le bouton 'évènements' affiche une liste de choix : texte, bulle, image, boucle, modèle 3D... Fig.3.
Chaque évènement possède une série d'options et de configurations
Enfin, la sauvegarde est effectuée en temps réel et vos vidéos seront hébergées par la fondation Mozilla gratuitement Fig.4.

Fig.1

Fig.3

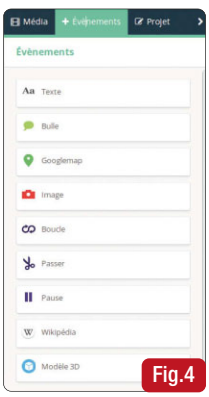


Fig.4

En pratique

Tout d'abord, nous insérons dans différents calques le logo Programmez et un texte qui sera affiché avec un effet de fondu. Il sera ensuite affiché une petite image, comme un elePHPant.
La particularité de l'introduction, est l'image car le logo sera une image et l'image de l'éléPHPant sera la dernière image taguée elePHPant venant du site Internet flickr, c'est à dire qu'elle changera à chaque fois que ce site sera mis à jour.
L'étape suivante sera la vidéo, nous avons retenu 2 vidéos et nous utiliserons 2 calques supplémentaires pour en afficher une partie. Par ailleurs, nous ajoutons un petit lien qui sera affiché pendant une courte durée. Enfin, nous ajoutons en bas à gauche un slideshow avec les images taguées #elePHPant de Flickr.
L'étape de fin sera l'apparition d'un logo et du lien pour accéder à un site Web que nous affichons quelques secondes.
Il est important d'enregistrer, et nous obtenons un lien que nous pouvons partager :
https://hellosct1.makes.org/popcorn/2wyw

Plus

L'ensemble des fonctions ne peuvent pas être présentées ici, mais il ne faut pas hésiter à les essayer. Par ailleurs une aide est disponible pour vous aider à obtenir plus d'informations. Vous pouvez créer directement votre propre version à partir du code source mis à disposition sur GitHub (https://github.com/mozilla/popcorn.webmaker.org).

X-ray

X-Ray est une application intéressante car elle va regarder en profondeur les différents niveaux de vos pages Web sur le principe de lunettes à rayons X. Ainsi, vous pouvez remixer un contenu en un simple clic, en y ajoutant vos propres textes, images...
Vous allez pouvoir habiller à votre manière la page d'accueil d'un de vos sites favoris comme un site d'actualité ou de média. De plus, vous êtes libre de l'habiller comme vous en avez envie en modifiant les couleurs ou en le transformant en site humoristique...
Son rôle n'est pas de hacker un site mais de comprendre comment fonctionne un bloc ou une partie d'une page Web. Le principe est de cloner le site Internet vers votre espace d'hébergement et en ajoutant vos personnalisations.

En pratique

X-Ray s'utilise de différentes manières et à partir d'un addon de Firefox que l'on place dans la barre de navigation (https://webmaker.org/fr/goggles/install). Après l'installation de celui-ci, il est tout de suite opérationnel Fig.5.
Pour utiliser X-Ray, il vous suffit de naviguer sur Internet et si une page d'accueil vous intéresse, vous pouvez l'utiliser. Nous prendrons pour exemple le site du magazine Programmez que vous connaissez Fig.6.
Nous allons déplacer le logo 'Programmez' sur la droite à la place de se connecter et ajouter un autre logo. Pour réaliser cette opération,

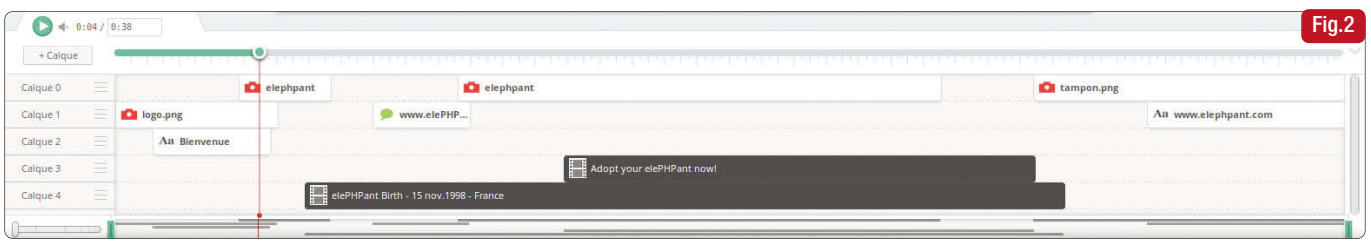


Fig.2

nous activons X-ray et sélectionnons le logo de la page Fig.7.
Une nouvelle fenêtre apparaît avec le contenu de l'élément que nous avons sélectionné Fig.8.
A ce moment là, nous pouvons changer l'image par une autre. Nous effectuons la même opération avec le mot de 'se connecter' que nous remplaçons par le logo d'origine Fig.9.
Comme l'image n'est plus proportionnée, il est possible de modifier la feuille de style avec X-Ray. Pour cela, nous sélectionnons l'objet et appuyons sur la touche C pour voir les informations CSS de notre élément Fig.10.
Pour modifier les valeurs, il faut appuyer sur la touche 'espace'.
Lorsque les modifications ont été effectuées, nous sauvegardons.
La dernière étape dans les modifications du bandeau, concerne le texte « s'inscrire » dont nous modifions aussi bien la couleur que le texte Fig.11.
Enfin, pour publier, il suffit d'appuyer sur la touche P pour publier et nous obtenons le résultat final qui suit : Fig.12.
Enfin, une nouvelle URL est proposée que vous pouvez sauvegarder et réutiliser sans problème.

Thimble

Thimble est une application pour créer et partager votre propre page Web. Vous allez pouvoir écrire et modifier le code HTML et CSS d'une page directement à partir de votre navigateur de préférence Firefox en un clin d'oeil. Par ailleurs, vous n'aurez aucun soucis pour héberger vos sites Web car tout peut se faire directement sur place, quel que soit votre niveau de connaissance (débutant ou experts).
Les réalisations possibles sont variées car vous pouvez créer une carte personnalisée, une affiche ou des pages délirantes, tout en étant là pour être assisté.

En pratique

Le remix est souvent la première étape pour apprendre le Web, car comme son nom l'indique, il remixe les pages des autres pour vous aider à créer les vôtres.
Pour exécuter l'opération, nous prenons un des sites disponibles sur Webmaker (https://webmaker.org) et cliquons sur le bouton 'remixer'
L'écran se divise en 2 parties :

- La partie de droite est l'aperçu de la page qui a été créée ;

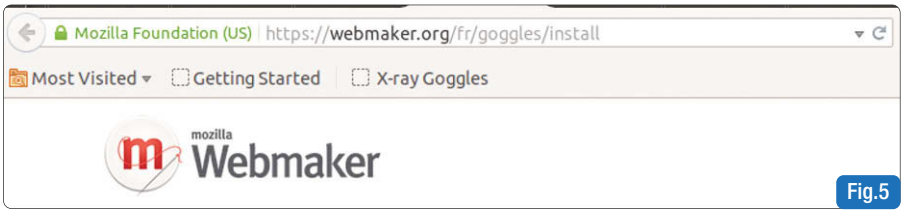


Fig.5

Fig.6

Fig.7

Fig.8

Fig.9

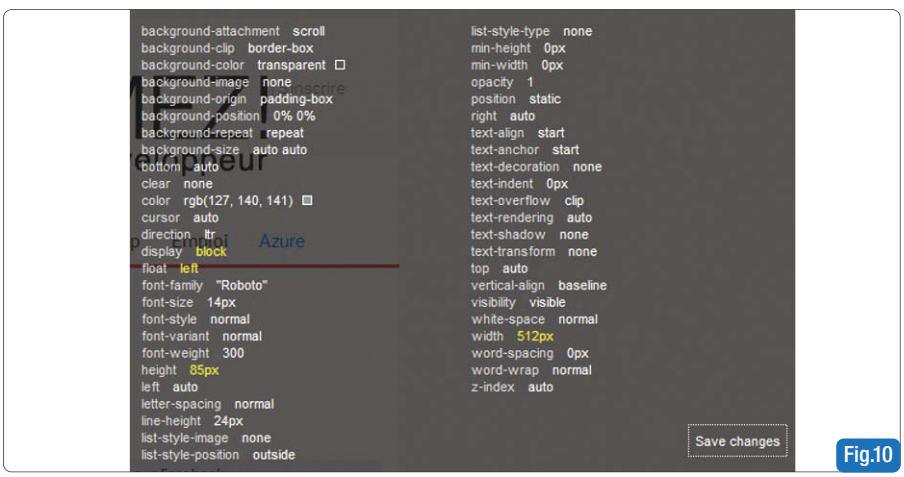


Fig.10

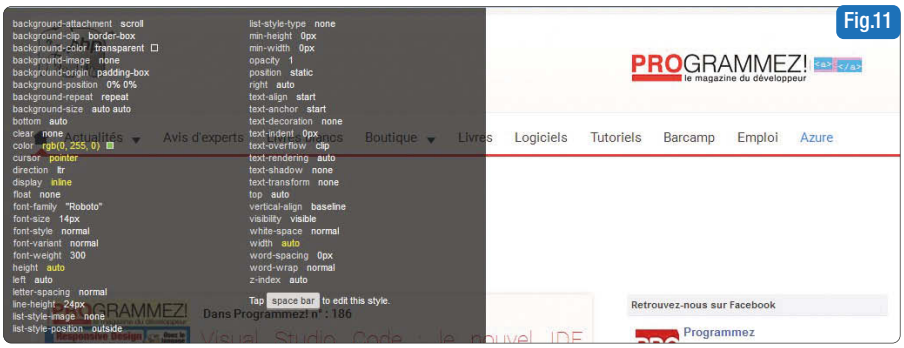


Fig.11

Fig.12

- La partie de gauche est le code HTML / CSS et correspond à l'assemblage de la partie de droite (visuel) : Fig.13.

Nous choisissons de remplacer le mot 'description' et son texte Fig.14.
Le résultat est instantané et affiche en temps réel les modifications que vous apportez. Enfin, il est important de sauvegarder les modifications pour obtenir le nouveau lien.
Pour aller plus loin :
https://support.mozilla.org/fr/products/webmaker/thimble

Appmaker

Il s'agit du dernier outil de la famille de la plateforme Webmaker. Il va vous aider à créer des applications mobiles, même si vous ne savez pas coder Fig.15.
De nombreux composants sont disponibles pour créer et partager les applications mobiles depuis votre navigateur. De nombreux groupes d'objets sont disponibles comme des composants média, fun, chat, numéro, etc.
Ainsi, vous pouvez créer :

- Une application de tchat pour parler avec vos amis ;
- Une application pour faire de la musique ;
- ...

En pratique

Nous allons vous proposer un compteur automatique, c'est à dire que nous choisissons le composant 'nombre' et un bouton qui servira de 'validation' Fig.16.
Chacun d'eux est modifiable.

Quand l'application est terminée, nous sauvegardons et publions l'application. Ainsi différentes possibilités vous sont proposées :

- L'installation,
- L'exécution,
- Envoyer un lien (Fig.17).

L'application est fonctionnelle et le chiffre augmente de 1 quand on clique sur le bouton. Fig.18.

EN RÉSUMÉ

Les outils Webmaker de la fondation Mozilla sont gratuits. De plus, vous pouvez afficher la source et voir le contenu de vos différents projets : vous êtes 100 % propriétaire de vos applications et réalisations. Notez que de nombreux évènements ont lieu en France pour apprendre le Web facilement et qu'ici, cet ensemble d'outils qui vous est proposé pour comprendre comment fonctionne une page Web s'avère très complémentaire, voire, peut se substituer, tout du moins au début.

Ressources :

https://webmaker.org/fr/tools
https://support.mozilla.org/fr/products/webmaker

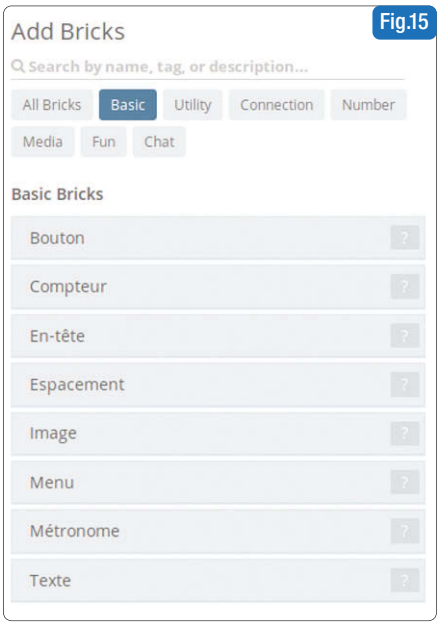


Fig.15

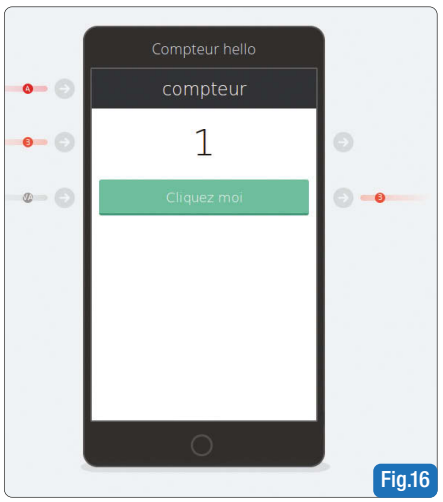


Fig.16

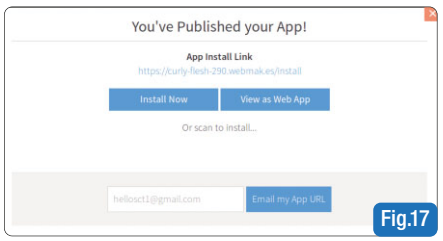


Fig.17

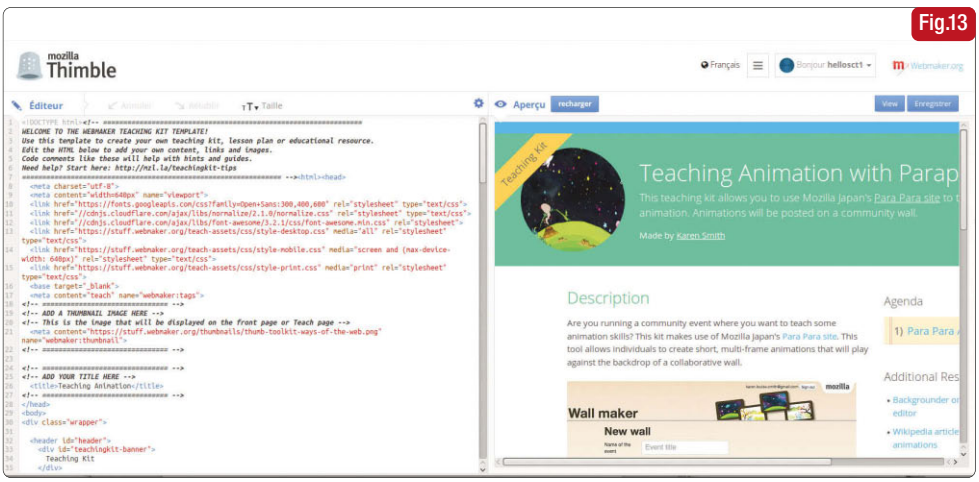


Fig.13

```
<!-- ================================ -->
<!-- DESCRIPTION -->
      <section>
        <h1>Programmez</h1>
        <div>
          <p>Le magazine du développeur<br><br>  <hr><br><br></p>
        </div>
      </section>
<!-- ================================ -->
```
Fig.14

Fig.18

Web Workers : le multithread débarque dans le navigateur !

Proposant une expérience utilisateur toujours plus riche, les applications Web sont écrites en Javascript. Alors que leurs fonctionnalités les rapprochent des applications natives, elles souffrent depuis toujours d'une limite de taille liée au modèle d'exécution monothread du Javascript au sein du navigateur. Ainsi, les opérations lourdes réalisées côté client se traduisent souvent en message d'avertissement du navigateur conseillant de tuer le script en question, ce qui est du plus mauvais effet. Avec les Web Workers, HTML 5 embarque une spécification apportant le multithread directement au sein du navigateur. Découverte d'une technologie prometteuse qui va accélérer encore un peu plus les applications Web.


Sylvain SAUREL
Ingénieur d'Etudes Java / Java EE
sylvain.saurel@gmail.com

Né au milieu des années 90, le langage Javascript est rapidement devenu le compagnon indispensable d'HTML. A cette époque, les ordinateurs étaient dotés de processeurs simple cœur et son modèle d'exécution basé sur un thread unique se prêtait plutôt bien à ses usages. En effet, les sites de l'époque étaient à des années lumière de qui se fait aujourd'hui avec des possibilités restreintes comparées aux applications natives.

Ainsi, l'exécution d'un script n'engage qu'un seul thread au niveau du navigateur. Plus connu sous le nom d'UI Thread, il est également en charge de l'affichage et du rendu. Avec la complexification des applications Web, les traitements Javascript n'ont cessé de s'alourdir conduisant quelquefois à des IHM bloquées. Les navigateurs disposent d'un système de protection permettant d'avertir l'utilisateur lorsque l'exécution d'un script est trop longue lui proposant de le tuer (Fig.1) ce qui se révèle désastreux en termes d'expérience utilisateur.

Néanmoins, jusqu'alors, les scénarios d'utilisation où cela posait de réels problèmes étaient limités, bien que les applications Web avec leurs contenus toujours plus riches commençaient à être pénalisées. Pour dépasser ces limitations historiques, les développeurs avaient trouvé des parades permettant de simuler des traitements parallèles. Elles étaient basées sur les fonctions setTimeout / setInterval, l'utilisation de l'objet XMLHttpRequest pour les appels HTTP asynchrones ou encore le système évènementiel du DOM; ces approches étaient non bloquantes mais n'apportaient rien au niveau de l'accélération des traitements. Alors que la norme est aux ordinateurs multi-cœurs et qu'HTML 5 propose des possibilités toujours plus puissantes, la présence du multithread dans Javascript est une nécessité indispensable aujourd'hui, et plus encore pour les années à venir.

Un peu de théorie

Afin d'adresser cette problématique, le W3C a intégré au sein d'HTML 5 la spécification Web Worker qui propose une API pour exécuter des scripts Javascript en tâche de fond. Il devient alors possible de créer des threads séparés à partir de l'UI Thread d'une application Web. Pour éviter les problèmes inhérents aux partages de ressources entre threads, tels que les locks ou les races conditions, le cadre d'exécution des Web Workers est une sandbox relativement stricte au niveau des ressources accessibles. Ils ne peuvent ainsi pas accéder aux objets Javascript standards comme

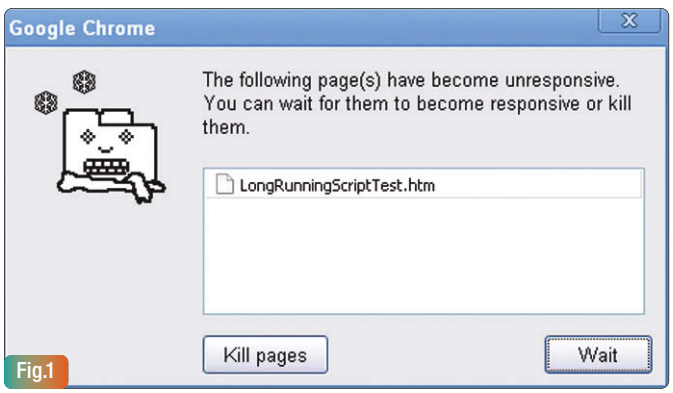


Fig.1
Avertissement script trop long

document, window, console, parent et également le DOM. Impossible de ce fait d'accéder à un élément via un getElementById ou d'utiliser la fonction bloquante alert.

Ces restrictions risquent de frustrer les développeurs habitués au multithreading sur des environnements natifs, mais elles permettent de préserver les développeurs Web des nombreux écueils du monde parallèle. Malgré ces limitations, les possibilités sont nombreuses et le développeur peut interagir avec les éléments suivants dans un Web Worker :

- L'objet navigator ;
- L'objet location en lecture seule ;
- La fonction importScripts() pour importer des scripts issus du même domaine ;
- Les objets Javascript de base : Object, Array, Date, Math et String ;
- L'objet XMLHttpRequest pour les requêtes HTTP ;
- Les fonctions setTimeout et setInterval ;
- La base de données Indexed DB ;
- L'objet self représentant les objets dans le scope du worker courant.

La spécification introduit 3 types de Web Workers. Le Dedicated Worker correspond à un nouveau thread exécuté en arrière-plan ne bloquant donc pas le travail de l'UI Thread. Associé à ce type, le Sub Worker est un Dedicated Worker un peu particulier puisque créé au sein d'un autre worker. Enfin, on retrouve le Shared Worker utilisable par plusieurs pages via des connexions multiples sur le même domaine d'origine.

Communication et gestion des erreurs

Pour conclure cette introduction théorique aux Web Workers, il est important de parler du mode de communication retenu entre les workers et le thread principal. Implémenté sous la forme de messages, il est réalisé via la fonction postMessage sur un worker depuis l'UI Thread ou directement à l'intérieur du worker pour envoyer un message à l'UI Thread. La fonction postMessage prend en entrée n'importe lequel des objets primitifs du Javascript, et également des objets JSON. Au sein du script associé à un worker, il faut s'abonner à l'évènement message avec un callback appelé

lors des passages de messages depuis l'UI Thread. Le travail du thread démarre ainsi au premier appel à la fonction postMessage depuis la page principale.
Une fois un worker terminé, sa fermeture reste à la charge du développeur. Compte tenu de la mémoire prise au sein du navigateur, il est important de réaliser cette opération. Pour cela, on peut appeler la méthode terminate sur l'instance de worker au sein de la page principale ou bien directement clore le worker dans son script via un appel à self.close().
Au niveau de la gestion des erreurs, le worker propose l'évènement error auquel il faudra s'abonner pour récupérer les informations détaillées associées à une erreur du code d'un worker. L'objet ErrorEvent passé au callback de cet évènement propose le nom du worker, le message d'erreur remonté ainsi que la ligne correspondante au sein du script.

Cas d'utilisation

Avec les applications Web actuelles nécessitant des interfaces utilisateurs riches et réactives, les scénarios d'utilisation pour la mise en place des Web Workers sont nombreux. Bien que leurs restrictions réduisent leur portée, on peut citer les cas suivants :

- Traitement de gros volumes de données. Remontés via XMLHttpRequest, ces volumes peuvent être traités au sein de workers soulageant par là-même l'UI Thread ;
- Traitement d'image. Les possibilités offertes par le Canvas ou la balise video apportent de nouveaux cas d'utilisation nécessitant une grande puissance de calcul. La parallélisation de ces calculs est une option avantageuse ;
- Traitements lourds. Que ce soit pour de l'analyse de texte au sein de suites bureautiques en ligne, ou bien pour réaliser des calculs financiers poussés sur des sites boursiers, les workers auront leur mot à dire ;
- Utilisation concurrente de la base de données Indexed DB. Si l'accès aux données du cache local n'est pas autorisé dans les workers, la base de données Indexed DB est totalement opérationnelle ;
- Jeux. Les moteurs physiques ou bien les IA de jeux HTML 5 sont également d'excellents candidats.

Aussi alléchants que soient ces scénarios, il est bon de garder à l'esprit que la programmation parallèle est une problématique complexe nécessitant du pragmatisme. Ainsi, tous les traitements ne pourront pas tirer parti des Web Workers. En sus, la création d'un worker est une opération coûteuse qui peut prendre plus de temps que le gain réalisé à l'exécution du traitement via un worker pour des calculs trop modestes. Toujours rayon performance, le coût mémoire supplémentaire induit devra être soigneusement quantifié.

Premier Worker

Pour mettre en œuvre un premier Dedicated Worker, nous allons prendre l'exemple classique de la recherche de nombres premiers. Une telle recherche est un traitement plutôt lourd qui bloquerait l'UI Thread et rendrait donc inutilisable l'IHM pour l'utilisateur. L'utilisation d'un worker est tout à fait à propos pour éviter ce phénomène. Mieux encore, ce traitement se prête parfaitement à la parallélisation puisque chaque recherche de nombre premier est indépendante de la précédente. Nous pouvons donc créer un script pour le worker nommé prime.js prenant en entrée une plage de nombres pour laquelle la primalité sera testée :

```
self.onmessage = function messageHandler(event) {
 var wid = event.data.wid;
 var n = event.data.start;
 var threshold = event.data.end - event.data.start;
 var total = 0;
 var found = 0;

 while (total < threshold) {
  n += 1;
  var max = Math.sqrt(n);
  var ok = true;

  for (var i = 2; i <= max  &&  ok; i += 1) {
   if (n % i == 0) {
    ok = false;
   }
  }

  if (ok) {
   found++;
  }

  total++;

  if (total % event.data.step == 0) {
   this.postMessage({wid : wid, total : total, found : found});
  }
 }
}
```

On s'abonne à l'évènement message en plaçant un callback sur la propriété self.onmessage. Le nombre d'entiers à tester est défini dans la variable threshold en calculant la différence entre les bornes de début et de fin. Enfin, on stocke le nombre d'entiers premiers trouvés et le nombre d'entiers testés. Le paramètre step de l'objet JSON récupéré via event.data.step définit la fréquence à laquelle on doit avertir l'UI Thread de l'avancée de la recherche. La communication des données vers ce thread principal s'effectue via un appel à la fonction postMessage avec en entrée un objet JSON proposant également l'id du worker ayant envoyé le message.
Il reste maintenant à créer les différents workers au sein de la page principale en passant en entrée de chacun d'entre eux les plages de valeurs à tester :

```
var width = 0;
var percentage = 0;
var total = 0;
var found = 0;
var t = [];
var f = [];
var sum = function(a, b) { return a + b;};

function messageHandler(event) {
 t[event.data.wid] = event.data.total;
 f[event.data.wid] = event.data.found;
 total = t.reduce(sum);
 found = f.reduce(sum);

 // affichage UI
 var prime = document.getElementById('prime');
 prime.innerHTML = 'Premiers trouvés: ' + found;
 percentage = Math.round(total * 100 / 1000000);
 width = Math.round(310 * percentage / 100);
 var progressbar = document.getElementById('progressbar');
 progressbar.innerHTML = percentage + '%';
```

```
 progressbar.style.width = width + 'px';
}

function processFor(number, workers) {
 var block = number / workers;

 for (var i = 0; i < workers ; i += 1) {
  var worker = new Worker('prime.js');
  worker.onmessage = messageHandler;
  worker.postMessage({wid : i, start : i * block, end : (i + 1) * block, step : block / 10});
 }
}
```

La fonction processFor démarre la recherche d'entiers premiers avec un seuil défini en choisissant le nombre de workers à utiliser.
Chaque worker est ensuite créé au sein d'une boucle avant de lui passer en entrée la plage de valeurs.
La fréquence de mise à jour est primordiale ici puisqu'une mise à jour à chaque entier testé reviendrait à bloquer l'UI Thread ! Pas à cause du calcul des nombres premiers mais à cause de la mise à jour de l'affichage de l'UI réalisé au sein de la fonction messageHandler.
En effet, la recherche d'éléments de la page au sein du DOM pour leur mise à jour est une des opérations les plus coûteuses en Javascript. L'exécution du script sur 1 million d'entiers met en lumière l'efficacité des workers puisque l'UI reste parfaitement réactive (Fig.2).

Traitement d'image

Comme vu précédemment, les Web Workers se prêtent particulièrement bien au traitement d'images dans le navigateur, et c'est un usage qui va se répandre avec l'arrivée du composant Canvas et de l'élément video dans HTML 5. Pour réaliser un exemple de ce type, nous partons d'une image dessinée au sein d'un Canvas sur laquelle pourront être effectués des traitements d'image simples tels qu'un effet Sepia ou un effet de niveaux de gris. Travaillant pixel par pixel, ces traitements sont de fait de parfaits candidats à la parallélisation. Il suffit de découper l'image à traiter en sous parties qui seront traitées au sein de workers dédiés.
Un worker n'ayant pas accès à l'élément Canvas, il va être nécessaire de travailler directement sur les données binaires de l'image chargée, en découpant l'image en autant de régions que de workers à employer :

```
var canvas = document.getElementById("target");
canvas.width = source.clientWidth;
canvas.height = source.clientHeight;
var tempContext = canvas.getContext("2d");
var len = canvas.width * canvas.height * 4;
// img initiale
tempContext.drawImage(source, 0, 0, canvas.width, canvas.height);
// parallélisation
var workersCount = 4;
var finished = 0;
var segmentLength = len / workersCount;
var blockSize = canvas.height / workersCount;

for (var index = 0; index < workersCount; index++) {
 var worker = new Worker("pictureProcessor.js");
 worker.onmessage = onWorkEnded;
 var canvasData = tempContext.getImageData(0, blockSize * index, canvas.width, blockSize);
 worker.postMessage({ data: canvasData, index: index, length: segmentLength });
}
```

Suivant le nombre de workers paramétrés, les différentes régions de l'image à traiter sont définies et envoyées ensuite à chacun des workers via la fonction postMessage. En entrée de celle-ci, on retrouve ainsi les données binaires de la région de l'image à traiter ainsi que la longueur de cette région. Contenu dans le fichier pictureProcessor.js, le code du worker exécute le traitement d'image souhaité :

```
importScripts("tools.js");

self.onmessage = function (e) {
 var canvasData = e.data.data;
 var binaryData = canvasData.data;
 var l = e.data.length;
 var index = e.data.index;

 processSepia(binaryData, l);
 // processGrayscale(binaryData, l);

 this.postMessage({ result: canvasData, index: index });
};
```

A la réception du message, le worker se met en route appliquant le traitement d'image sur les données passées en entrée. On note ici l'appel de la fonction importScripts pour importer le script tools.js contenant les fonctions de traitement d'image en question. Une fois le traitement exécuté, le worker envoie un message à l'UI Thread contenant les données binaires transformées ainsi que l'index de la région concernée. Au niveau de la page principale, la réception de ce message est faite au sein du callback onWorkEnded :

```
var onWorkEnded = function (e) {
 var canvasData = e.data.result;
 var index = e.data.index;
 // img resultat
 tempContext.putImageData(canvasData, 0, blockSize * index);
 finished++;

 if (finished == workersCount) {
  // fin traitement
 }
};
```

Les données transformées sont affichées au sein du Canvas afin de présenter le résultat de la transformation à l'utilisateur. Un compteur permet

Calcul de nombres premiers via Worker

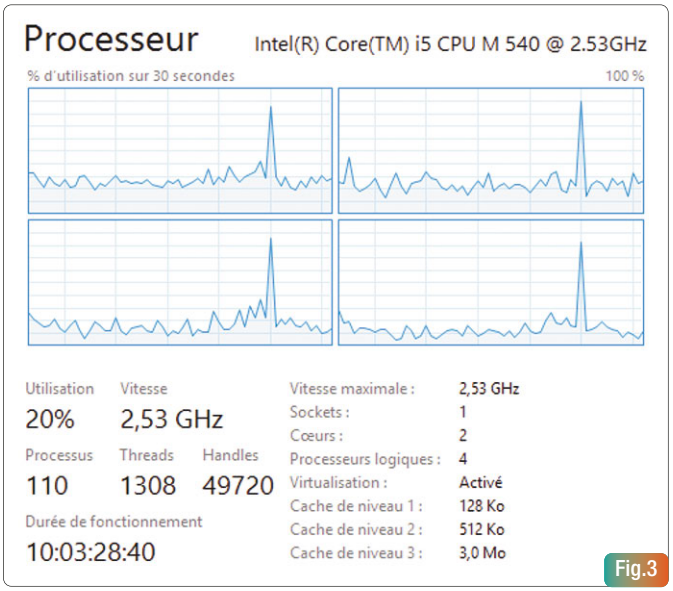


Fig.3

Exécution du script avec 4 Workers

de déterminer la fin d'exécution du traitement et de proposer éventuellement un retour à l'utilisateur. Non détaillé ici, le recours à la bibliothèque Javascript Modernizr et sa propriété Modernizr.webworkers offre la possibilité de proposer un traitement dégradé sans Web Workers dans le cas où le navigateur cible ne supporterait pas cette nouvelle fonctionnalité HTML 5. Non détaillées ici car n'étant pas l'objet premier de l'article, les fonctions de traitement d'image utilisées au sein de tools.js sont disponibles avec le code source des exemples de cet article. Enfin, l'exécution du script permet de constater que le navigateur utilise bien 4 threads pour traiter en parallèle l'image d'origine (Fig.3).

Shared Worker

Second type de workers, les Shared Workers peuvent être partagés entre pages issues d'un même domaine au sein du navigateur ce qui implique qu'il n'est pas rattaché uniquement au script qui l'a lancé. Ainsi, il peut recevoir plusieurs connexions. Outre cette différence d'implémentation fondamentale, les Shared Workers utilisent également le concept de port à partir duquel il faudra travailler. Le démarrage de l'écoute d'un worker de ce type se fait une fois l'appel à la méthode start de l'objet port réalisé. Fort logiquement, les variables globales définies au sein d'un Shared Worker sont accessibles lors des autres appels de ce worker. Enfin, la communication se fait également via postMessage mais appelé depuis l'objet port.
Un exemple simple illustrant le fonctionnement des Shared Workers consiste à mémoriser le nombre de pages s'y connectant et à renvoyer à la page appelante son numéro de connexion :

```
var num = 0;

self.onconnect = function (e) {
 var port = e.ports[0];
 port.postMessage("Connexion  #" + num);
 num++;

 port.onmessage = function (e) {
  // réponse ...
  port.postMessage("Réponse : " + e.data);
 };

 port.start();
};
```

Petite subtilité avec un Shared Worker, il est nécessaire de s'abonner à l'évènement de connexion connect pour ensuite se mettre en attente de réception de messages.
Au niveau de la page appelante, le démarrage du Shared Worker s'effectue aisément avec pour simple précaution de bien utiliser le concept de port lors des manipulations afférentes au worker :

```
var w = new SharedWorker("shared.js");
w.port.onmessage = function(e){
 // msg dans e.data
};

w.port.start();
w.port.postMessage("Bonjour");
```

L'exécution du script met en exergue le principe de fonctionnement du Shared Worker puisque le numéro de connexion est bien affiché au sein de la page appelante. L'existence du Shared Worker n'est pas liée à la page ou à l'UI Thread dont il est issu. En fait, il continue à tourner tant qu'au moins une page y étant connectée reste ouverte. Sa destruction s'effectue donc à la fermeture de la dernière page connectée ou bien lors de l'appel explicite aux fonctions close ou terminate selon que l'on soit au sein du worker ou de l'UI Thread.

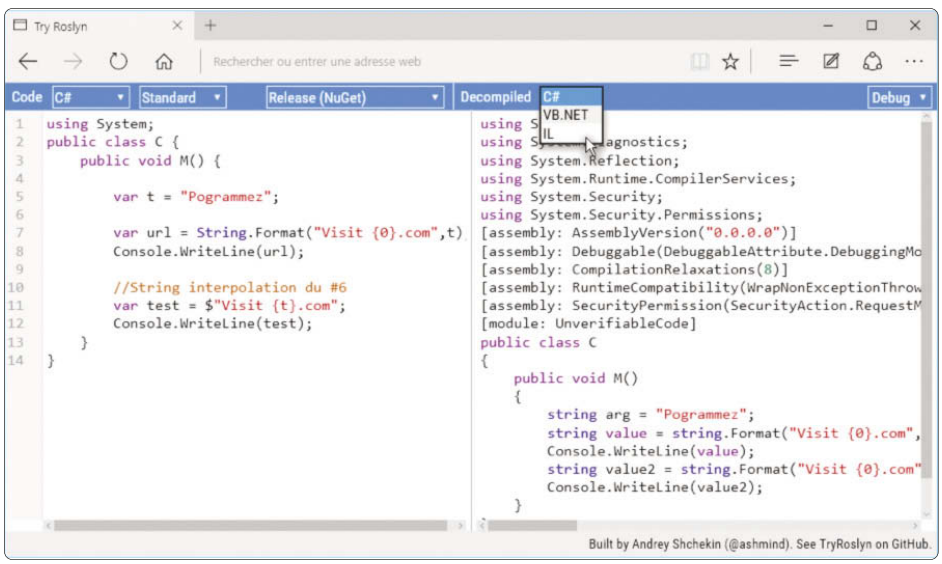
Conclusion

Comblant un manque historique de Javascript lié à son modèle d'exécution monothreadé, les Web Workers vont rendre les applications Web encore plus réactives. Les esprits le plus chagrins regretteront les limitations dues à leur exécution au sein d'une sandbox, mais c'est un choix d'implémentation logique au vu des écueils induits par la programmation parallèle. Les possibilités demeurent cependant assez importantes pour en tirer profit sur de nombreux cas d'utilisation. La facilité de manipulation de l'API associée aux Web Workers ne doit néanmoins pas perdre de vue que la parallélisation des traitements reste un domaine complexe pas forcément applicable à tous les cas de figure. Enfin, compte tenu de leur temps démarrage et de leur consommation mémoire, il sera nécessaire d'évaluer correctement si les gains apportés par leur emploi se révèlent bien supérieurs à ces temps de fonctionnement.

Essayez le compilateur Microsoft Roslyn en ligne

Microsoft "Roslyn" est un nouveau compilateur C# et VB open source de la plateforme .NET. Il introduit ce que l'on appellera « Compiler-as-a-service ». Pour la petite histoire, on voit souvent pas mal de projets Microsoft qui portent de drôles de dénominations. Et Roslyn est aussi un petit village de 900 habitants à 1h30 de Redmond. Peut-être que l'initiateur du projet y habite.

Christophe Peugnet
MVP Windows Plateform Devlopment chez SodeaSoft / EBLM
http://www.sodeasoft.com
Blog : http://www.peug.net
Twitter : @tossnet1



Habituellement on imagine un compilateur comme une sorte de machine hermétique, magique même, dans laquelle on lui injecte du code source qui est alors analysé, transformé et traduit tandis qu'en sortie on obtient notre applicatif quasi prêt à être utilisé par notre client.

Cette machine, cette boîte noire n'est plus. Grâce à la plateforme de compilation .NET "Roslyn", les développeurs ont maintenant accès aux informations du compilateur et à l'interprétation de leur code source par le compilateur. Cela crée de nouvelles opportunités pour l'innovation dans des domaines tels que Meta-programmation.

Roslyn offre donc au développeur C# ou Visual Basic un outil lui permettant d'analyser son code à travers une API exposant les services de compilation du Framework .NET

Pour les développeurs plus expérimentés, il existe, au sein de Visual Studio, un paquet NuGet "Roslyn" qui permettra par exemple d'analyser le code directement sous forme textuelle. Vous trouverez ce package NuGet via ce lien :
http://www.nuget.org/packages/Microsoft.CodeAnalysis

Ce compilateur étant open source, j'ai découvert il y a peu qu'un développeur, Andrey Shchekin (@ashmind), a créé un site Web où vous pouvez tester Roslyn... et attention on y devient addict ! On peut donc y tester les petites nouveautés du C#6, et se rendre compte immédiatement comment le compilateur interprète notre code
http://tryroslyn.azurewebsites.net/

Comme vous pouvez le voir sur la capture-écran, à gauche vous pouvez saisir votre code en C# ou en VB.NET. Différentes releases vous sont aussi proposées. Et sur la droite de l'écran, vous pourrez lire le résultat de Roslyn en C#, VB.NET ou encore en IL (Intermediate language) et ceux en mode Debug ou en mode Release. Vous constaterez que c'est très instructif !

MEAN.IO (3/4)

Voici le troisième article de la série consacrée à MEAN.IO, après la présentation générale du framework lors du premier et celles des concepts permettant de créer un nouveau module dans le second, nous nous attarderons ici sur un cas d'application concret : celui de la visualisation de données géographiques. Depuis le départ, nous souhaitons en effet créer une application permettant de visualiser un ou plusieurs itinéraires GPS (type randonnée VTT ou pédestre). Nous avons défini précédemment le modèle de données permettant de stocker l'information en base, ainsi que l'API REST permettant de le manipuler. Ensuite, nous avons créé une partie cliente incluant des interfaces homme-machine (IHM) pour la présentation et l'édition de ces données. Il ne nous reste donc maintenant qu'à ingérer et visualiser nos chemins sous formes de cartes à la façon de "Google Maps" (objet du présent article) ou de vues 3D à la façon de "Google Earth" (4e et dernier article de la série).

Luc CLAUSTRES
Consultant indépendant
Créateur d'Applications Numériques
http://www.digital-innovation.fr

MISE À JOUR DE L'APPLICATION

Partie serveur (back-end)

Mise à jour du modèle

Nous avions défini dans l'article précédent le schéma MongoDB de l'objet unique manipulé par notre application qui est un itinéraire GPS ('track' en anglais). Un tel chemin est simplement décrit par une liste de positions GPS acquises par le capteur. Chaque point est repéré en coordonnées géographiques : longitude, latitude et altitude. A part l'utilisateur qui l'a créé, un titre et un descriptif, le chemin contenait donc un tableau de ces coordonnées. Néanmoins, à ce stade, nous n'avons aucun moyen d'ingérer dans notre application le chemin qui a été suivi et que l'on souhaite représenter. Comme ce type de données est généralement acquis de façon automatisée (via un GPS) et trop volumineux pour être ingéré manuellement, nous allons créer différentes fonctions permettant de facilement remplir notre base de données avec ces informations.

Dans le domaine il existe de nombreux formats mais les plus usités concernant une trace GPS sont probablement ces deux-là (tous deux basés sur XML) :

- GPX (GPS eXchange Format), un format ouvert permettant de décrire une collection de points utilisables sous forme de chemin (waypoint), trace (track) ou itinéraire (route).
- KML (Keyhole Markup Language), un format basé sur COLLADA et popularisé par Google mais qui est aujourd'hui devenu un standard international.

Afin d'optimiser le stockage en base, de ces données, nous avons toutefois utilisé un tableau de coordonnées brutes au sein de notre modèle. Pour simplifier le code de conversion nous passerons par un format pivot vers lequel seront transformées les données d'entrée GPX/KML. Afin de disposer d'une API facilement utilisable par des librairies cartographiques, nous souhaitons également utiliser ce format pivot en sortie. Nous focaliserons ainsi nos efforts sur le code de conversion du format pivot vers/depuis notre base de données.

Le format pivot qui s'impose de lui-même pour un code JavaScript est GeoJSON (de l'anglais Geographic JSON), qui est un format ouvert d'encodage de données géographiques basé sur la norme JSON (JavaScript Object Notation). Il permet de décrire des données de type point, ligne, polygone, ainsi que des ensembles de ces types de données et d'y ajouter des attributs quelconques. Il est de plus supporté par la plupart des librairies traitant l'information géographique. Enfin, il existe le module Node.js togeojson qui permet de convertir un arbre XML au format KML/GPX en GeoJSON. Il suffit donc de coupler ce module à un module de parsing de flux XML tel que jsdom pour obtenir très simplement la conversion de nos données d'entrée. Etant donné que ces informations sont volatiles, nous utilisons naturellement un attribut virtuel de Mongoose (en écriture seulement) pour ce faire. Ainsi, une fois rajoutées les deux dépendances Node.js au **package.json** de notre module MEAN.IO nous insérons le code suivant dans notre modèle pour convertir les données d'entrée :

```
var togeojson = require('togeojson'),
  jsdom = require('jsdom').jsdom;

// Setter permettant de remplir le chemin à partir de données au format KML
TrackSchema.virtual('kml')
.set(function (data) {
 if ( data ) {
  var kml = jsdom(data);
  var geojson = togeojson.kml(kml);
  this.set( 'geojson', geojson );
 }
});
// Code identique pour gérer le format GPX
...
```

Le code de conversion repose lui-même sur un attribut virtuel permettant de stocker en base les données converties dans notre format pivot (i.e. GeoJSON). Etant donné que nous utilisons ce format également en sortie, cet attribut est cette fois en lecture/écriture. Si la conversion vers le GeoJSON est triviale, celle depuis le GeoJSON est un peu plus complexe selon les différents types de données d'entrée, nous n'aborderons donc ici que le cas le plus simple (voir le code de l'article pour l'exemple complet).

Trucs & Astuces : depuis la version 2.4 MongoDB supporte nativement le stockage de données au format GeoJSON (http://docs.mongodb.org/v2.6/reference/geojson/), il serait donc tout à fait possible de stocker directement l'objet GeoJSON pour simplifier la manipulation, mais en complexifiant la structure de données

Il ne nous reste plus qu'à permettre à l'utilisateur de fournir un fichier KML ou GPX afin d'alimenter notre base de données. Dans une application réelle, le fichier serait tout d'abord transféré sur le serveur puis traité côté serveur afin d'optimiser la bande passante et de s'affranchir des limites de taille. En effet par défaut Express configure une limite de 100 Kb pour la charge utile des requêtes JSON
(voir https://github.com/expressjs/body-parser).

Trucs & Astuces : Il est possible de configurer la taille par défaut via une instruction du type app.use(express.json({limit:'50mb'})); à l'initialisation d'Express, ce qui nécessite aujourd'hui de modifier le code de MEAN.IO pour se faire (voir https://github.com/linnovate/mean/issues/1169). Vous pouvez également utiliser des outils dédiés tels que le package MEAN.IO upload ou ng-file-upload couplés à multer qui se basent sur les *Form Data* pour disposer d'un transfert de fichier fiable.

Néanmoins, dans l'optique de simplifier notre exemple, nous allons lire le fichier côté client et envoyer directement les données lues avec la requête de création ou d'édition du chemin. Ainsi, seule la conversion en GeoJSON se réalisera côté serveur. Grâce aux attributs virtuels de notre modèle il suffit de rajouter une propriété kml ou gpx au contenu de notre requête pour que la magie opère ! Pour faire cela simplement j'ai créé une directive AngularJS (voir le code complet de l'article) qui à chaque changement dans le sélecteur de fichier de notre formulaire va récupérer le nom et le contenu du fichier sélectionné via la File API d'HTML5. Au niveau du contrôleur l'ajout du contenu du fichier dans la propriété adéquate de note modèle (kml ou gpx) se base sur l'extension du fichier lu :

```
...
  var track = $scope.track;
  // Ajout du contenu du fichier si présent
  if ( track.file.content ) {
    track[track.file.extension] = track.file.content;
  }
...
```

Afin de localiser plus simplement le chemin sur une carte nous allons également rajouter un champ contenant l'étendue géographique, traditionnellement nommé BBox (Bounding Box), car mathématiquement, il correspond à la plus petite "boîte" en deux dimensions qui englobe tous les points du chemin. Concrètement ce rectangle est défini par un point sud-ouest contenant la longitude et la latitude minimale, et un point nord-est contenant la longitude et la latitude maximale ; nous stockerons ces 4 valeurs sous la forme d'un tableau. Ceci nous permettra de recentrer la vue cartographique sur la zone concernée pour ne pas avoir à rechercher sur la carte. Au final le code du fichier **TrackModel.js** dans le dossier **models** du module est modifié comme suit pour le schéma et les fonctions d'importation/exportation des données :

```
// Déclaration du schéma du modèle 'Track'
var TrackSchema = new mongoose.Schema({
  ...
  // Boîte englobante des points (coordonnées géographiques également)
  bbox : {
    type : [Number], required : false
  }
});
// Getter permettant de récupérer le chemin au format GeoJSON
TrackSchema.virtual('geojson')
.get(function () {
  var coordinates = [];
  // En GeoJSON chaque point est un tableau alors que nous stockons à plat
  for (var i = 0; i < this.waypoints.length / 3; i++) {
    coordinates[i] = [ this.waypoints[3*i], this.waypoints[3*i+1], this.waypoints[3*i+2] ];
  }
  // Encapsulation des coordonnées dans le formalisme GeoJSON
  var feature = {
    "type": "Feature",
      "geometry": {
     "type": "LineString",
     "coordinates": coordinates
    }
  };
  return feature;
})
// Setter permettant de remplir le chemin à partir de données au format GeoJSON
.set(function (geojson) {
  if ( geojson && geojson.type == 'LineString' ) {
    // Récupération des coordonnées
    var coordinates = geojson.coordinates;
    // Transformation dans le formalisme de la base de données
    var waypoints = [];
    // Calcul de la boîte englobante
    var minLon = 360; var maxLon = -360;
    var minLat = 90; var maxLat = -90;
    for (var i = 0; i < coordinates.length; i++) {
      waypoints[3*i] = coordinates[i][0];
      waypoints[3*i+1] = coordinates[i][1];
      // Mise à jour des valeurs min/max
      minLon = Math.min(minLon, waypoints[3*i]);
      minLat = Math.min(minLat, waypoints[3*i+1]);
      maxLon = Math.max(maxLon, waypoints[3*i]);
      maxLat = Math.max(maxLat, waypoints[3*i+1]);
      // Nous nous assurons d'avoir des coordonnées 3D (pas requis en GeoJSON)
      if ( coordinates[i].length >= 3 ) {
        waypoints[3*i+2] = coordinates[i][2];
      } else {
        waypoints[3*i+2] = 0;
      }
    }
    this.set( 'waypoints', waypoints );
    this.set( 'bbox', [minLon, minLat, maxLon, maxLat] );
  }
});
```

Partie cliente (front-end)

Mise à jour des routes

Aux routes côté front-end, qui sont gérées via l'AngularUI Router, nous en ajoutons une nouvelle dans le fichier **ApplicationRoutes.js** du dossier **routes** de la partie publique, il s'agit de la déclaration permettant d'accéder à la page pour afficher la carte d'un chemin :

```
  // Page permettant de voir un chemin sur la carte
  $stateProvider.state('track map', {
    url: '/track/:trackId/map',
    templateUrl: '/application/views/TrackMap.html',
    controller: 'TrackMapController',
    resolve: {
```

```
    loggedin: function(MeanUser) {
      return MeanUser.checkLoggedin();
    }
  }
});
```

Mise à jour des vues

Pour la présentation des chemins (sous forme de liste ou sous forme unitaire) nous avions utilisé précédemment des panels Bootstrap. L'en-tête (classe CSS *panel-heading*) contenait notamment les actions réalisables par l'utilisateur (e.g. éditer ou effacer) sous forme d'icônes, auxquelles nous rajoutons la possibilité d'ouvrir la vue cartographique (**Fig.1**) dans **public/views/Track.html** :

```
<!-- Actions associés au chemin (effacer, éditer, vue 3D et vue carte) -->
<a data-ng-click="remove(track)"><i class="pull-right glyphicon glyphicon-trash"> </i></a>
<a href="/track/{{track._id}}/edit"><i class="pull-right glyphicon glyphicon-edit"> </i></a>
<a href="/track/{{track._id}}/map"><i class="pull-right glyphicon glyphicon-map-marker"> </i></a>
</div>
```

VUE CARTOGRAPHIQUE

La gestion de données cartographiques est un domaine qui nécessite un travail algorithmique important, afin d'assurer des temps d'accès rapides, tant le volume de données peut être conséquent. Imaginez par exemple que l'on couvre aujourd'hui la terre entière avec des images satellite à une résolution de quelques dizaines de centimètres. Une telle image sur une zone de 20 kilomètres carrés a une taille qui avoisine le milliard de pixels et pèse plusieurs giga-octets même compressée. Accéder à une base de données mondiale de ce type nécessite de pouvoir naviguer à travers des dizaines de téraoctets de données. Heureusement, des services mettant à disposition de telles données sont aujourd'hui accessibles gratuitement sur Internet, comme par exemple OpenStreetMap que nous utiliserons pour nous fournir un fond de carte sur lequel nous viendrons afficher nos tracés GPS.

Les fonds de carte de ce type sont généralement découpés en tuiles (i.e. images) de petite taille et décomposés sur plusieurs niveaux géographiques (échelles ou niveaux de résolution). L'idée générale pour obtenir un affichage fluide est, en fonction de la zone visualisée par l'utilisateur, d'identifier et de télécharger uniquement les tuiles visibles et dont la résolution est la plus adaptée à l'écran. Les tuiles présentent en général un grand nombre de niveaux de zoom, de la vue mondiale au détail de la rue, mais leur nombre à l'écran dépasse rarement quelques dizaines (**Fig.2**).

Fig.1

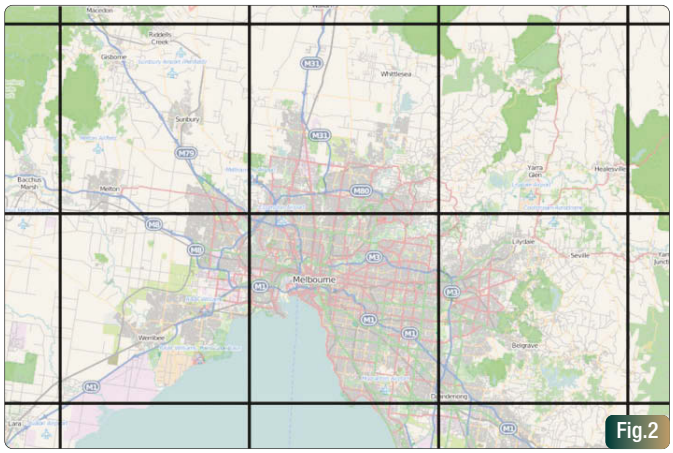


Fig.2

Pour la visualisation de données cartographique, les deux librairies Open Source les plus connues sont probablement à ce jour OpenLayers et Leaflet. David Rubert a eu la bonne idée d'initier des projets Open Source (auxquels j'essaye de contribuer) pour encapsuler ces deux librairies via des directives AngularJS, il s'agit de : angular-openlayers-directive et angular-leaflet-directive. Nous allons utiliser cette dernière pour notre application.

Directive

Pour ce faire, il faut rajouter "angular-leaflet-directive": "latest" dans le fichier bower.json à la racine de votre dossier applicatif et exécuter bower install. Ensuite il faut modifier le **app.js** du module applicatif pour rajouter les fichiers nécessaires à l'agrégation (voir article précédent) :

```
// Dépendances AngularJS du module
Application.angularDependencies(['mean.system', 'mean.users', 'leaflet-directive']);
Application.aggregateAsset('css', '../../../../../../bower_components/leaflet/dist/leaflet.css');
// La priorité permet de s'assurer que la librairie est chargée avant la directive
Application.aggregateAsset('js', '../../../../../../bower_components/leaflet/dist/leaflet.js', {weight: -1});
Application.aggregateAsset('js', '../../../../../../bower_components/angular-leaflet-directive/dist/
angular-leaflet-directive.js');
```

La directive se configure principalement via les attributs suivants :

- **defaults** : objet définissant la configuration de la carte (e.g. contrôles actifs) ;
- **center** : objet contenant les coordonnées du point central par défaut ainsi que le niveau de zoom ;
- **layers** : liste des couches disponibles pour le fond cartographique ;
- **markers** : liste de marqueurs (i.e. icônes) disposés sur la carte ;
- **geojson** : une couche de données au format GeoJSON affichée en surimpression sur la carte.

Notre configuration se contente d'autoriser le zoom avant/arrière via la molette et d'afficher également des boutons +/- pour se faire :

```
$scope.defaults = {
  zoomControlPosition: 'topleft',
  scrollWheelZoom: true
}
```

Chaque couche de données est caractérisée par un nom, un type (i.e. format) et une URL d'accès. Les différentes couches de fond cartographique sont sélectionnables via un menu intégré à la carte (en haut à droite **Fig.3**). Par exemple la couche de base proposée par OpenStreetMap est configurée comme suit :

```
$scope.layers = {
  baselayers: [{
    name: 'OpenStreetMap',
    type: 'xyz',
    url: 'http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
    layerOptions: {
      subdomains: ['a', 'b', 'c'],
      attribution: '© OpenStreetMap contributors',
```

```
    continuousWorld: true
  }]
}
```

Nous utiliserons la couche des marqueurs pour ajouter un simple marqueur indiquant le point de départ du chemin en indiquant ses coordonnées et un message apparaissant sous forme de bulle d'information :

```
$scope.markers.you = {
  lat: track.waypoints[1],
  lng: track.waypoints[0],
  message: "You are here"
}
```

Contrôleur

Le contrôleur de la vue cartographique récupère tout d'abord l'ID du chemin à visualiser dans l'URL grâce à la fonction findOne(). Ensuite il configure la couche de données au format GeoJSON à partir du résultat de la requête dédiée sur notre API (voir article précédent). La dernière instruction permet de recentrer la carte sur l'étendue géographique du chemin afin d'obtenir le résultat de la **Fig.3** :

```
// Contrôleur utilisé pour afficher un chemin sur une carte
angular.module('mean.application').controller('TrackMapController', ['$scope', '$http',
'$stateParams', 'TrackService', 'leafletData',
 function($scope, $http, $stateParams, TrackService, leafletData) {
   // Données chargées de façon asynchrone
   $scope.geoJSON = {};
   $scope.markers = {};
   //Récupère un chemin via son ID
   $scope.findOne = function() {
    TrackService.get({
      trackId: $stateParams.trackId
    }, function(track) {
      $scope.track = track;
      // Récupération des données au format GeoJSON
      $http.get('/api/track.GeoJSON/' + track._id).success(function(data, status) {
        angular.extend($scope, {
          geoJSON: {
            data: data,
            style: { weight: 5, color: 'red' }
          }
        });
      });
      leafletData.getMap().then(function(map) {
       // Zoom sur le chemin
       map.fitBounds( [ [track.bbox[1], track.bbox[0]], [track.bbox[3], track.bbox[2]] ] );
      });
    });
   };
  }
]);
```

Vue

La vue se contente d'instancier la directive et de binder les variables appropriées du scope sur les attributs dédiés :

```
<div data-ng-init="findOne()">
  <!-- Ajout d'une carte Leaflet -->
  <leaflet center="center" layers="layers" geojson="geoJSON" defaults="defaults" markers
="markers" width="100%" height="512px"/>
</div>
```

Conclusion

Cet article a été l'occasion de découvrir le domaine de la cartographie et d'approfondir votre connaissance du framework MEAN.IO en le mettant en pratique sur un cas concret d'utilisation via l'intégration de librairies externes. Lors du prochain épisode nous terminerons en beauté avec la visualisation de nos chemins en vue 3D à la façon "Google Earth".

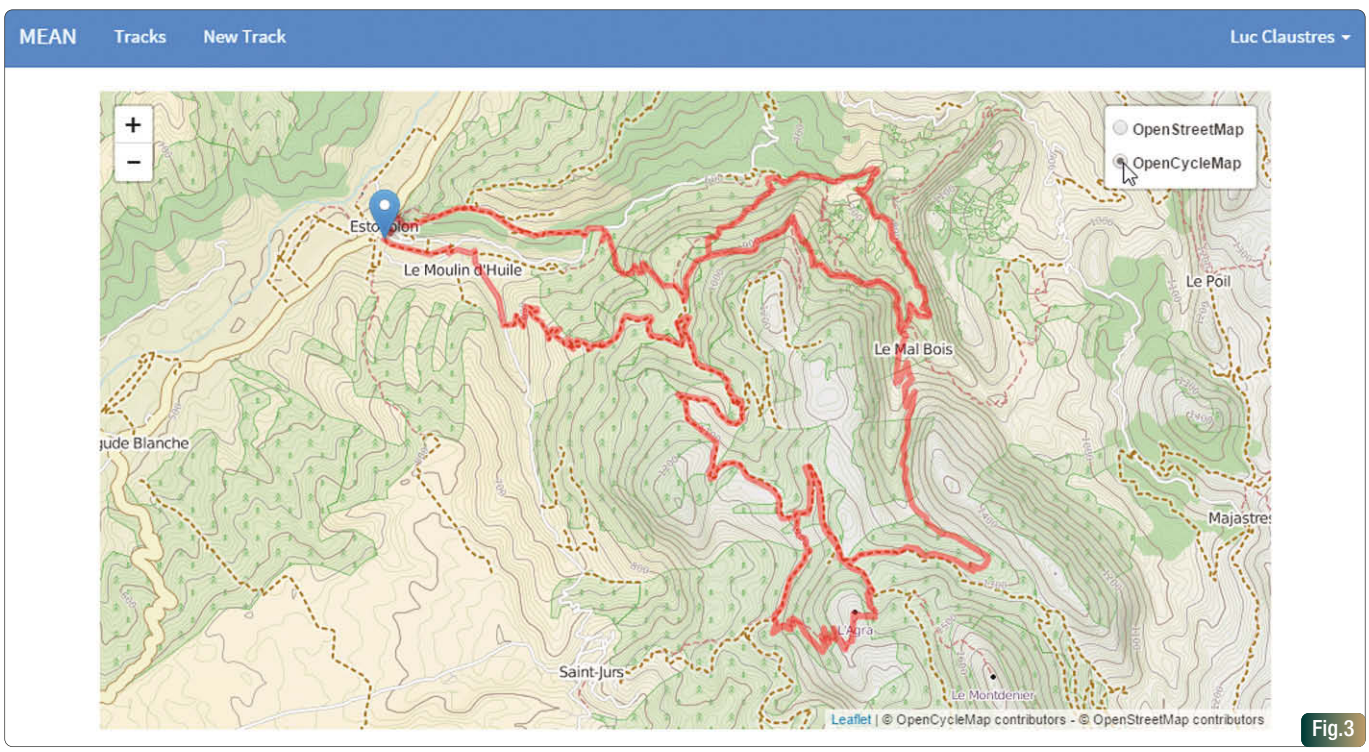


Fig.3

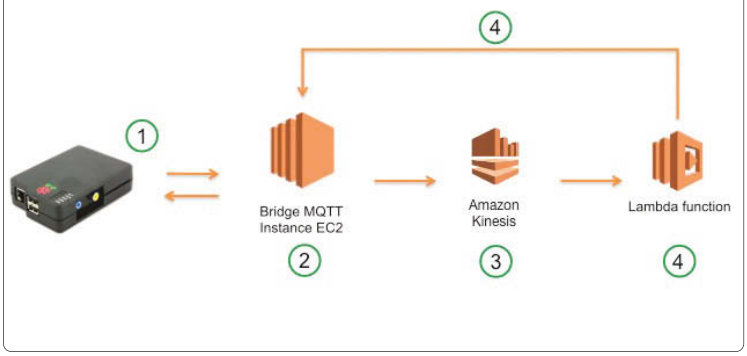
Créez des objets connectés avec le Cloud AWS !

Afin de réduire leurs coûts opérationnels, de construire de nouvelles lignes métiers, ainsi que d'augmenter la satisfaction et la fidélité de leurs clients, les entreprises parient aujourd'hui sur des applications reposant sur des objets connectés. Cette tendance fait accroître considérablement le nombre de ces objets connectés.

Michael Garcia
Amazon Web Services

Dans cet article, nous allons voir comment connecter des objets au Cloud AWS afin de les rendre intelligents pour vos utilisateurs. Vous serez ainsi en mesure de créer des applications à forte valeur ajoutée basées sur les données envoyées par les objets connectés, tout en ayant la possibilité de les commander à distance.
Pour créer cette forte valeur ajoutée vous pourrez bénéficier de l'étendue de la plateforme AWS pour rajouter des composantes mobiles, analytiques ou encore de machine learning à votre application.

Fig.1



Objets connectés et protocoles adaptés

Un large choix d'objets connectés existe que nous pouvons classer en deux catégories :

- Les objets connectés qui n'ont pas de limitation particulière ; leurs ressources peuvent leur permettre de communiquer en HTTP(s) sans considérations particulières vis à vis de la bande passante et ils sont capables d'exécuter du code écrit dans la plupart des langages de programmation actuels.
- Les objets dits « contraints » car ils possèdent au moins une limitation sur une de leurs ressources. Cela peut être une limitation au niveau de la puissance de calcul, de la bande passante, de l'autonomie ou encore de la mémoire disponible sur l'objet.

Pour répondre aux limitations des objets dits contraints qui ne peuvent pas se permettre de communiquer via HTTP(s), plusieurs protocoles ont été créés. Parmi plusieurs candidats nous avons choisi, de manière arbitraire, d'utiliser dans cet article le protocole MQTT qui est adapté aux objets très contraints de classe 1 : les objets de classe 1 possèdent environ 100 Kb de stockage, 10 Kb de RAM et une bande passante disponible faible. Le choix du protocole doit s'effectuer en fonction de votre cas d'usage et du type d'objet connecté utilisé.
Pour émuler un objet contraint nous allons utiliser un Raspberry Pi en raison de sa popularité et de son accessibilité afin de faciliter le prototypage pour les lecteurs. Dans un cas réel vous pourriez remplacer le Raspberry Pi par un micro contrôleur ou tout autre objet contraint.

Schéma d'architecture de la solution

Voici les fonctionnalités du prototype présenté dans cet article :

- Envoi de données à partir de l'objet connecté ;
- Traitement de ces informations ;
- Envoi de commande à l'objet connecté.

Nous allons maintenant détailler dans la suite de cet article chacune des parties de cette architecture (Fig.1) afin de comprendre le rôle de chacun des services AWS utilisés.

Etape 1 : Envoi et réception de messages sur un objet connecté

L'objet connecté doit être capable d'émettre des messages et de recevoir des commandes. Le protocole MQTT est un protocole basé sur un système de publication/notification qui est géré par un serveur (appelé « broker ») auquel tous les clients doivent se connecter. Une fois connectés les clients peuvent publier des messages sur un « topic », tous les clients qui ont souscrit à ce même topic reçoivent alors le message.
Nous allons utiliser le client open source Mosquitto, pour cela rendez-vous sur la page http://mosquitto.org/download/ et choisissez le client qui correspond à votre OS. Une fois installé vous pouvez tester la configuration avec un broker de test mis à disposition par mosquitto.org, pour cela ouvrez deux terminaux sur votre Raspberry Pi, dans la première fenêtre abonnez-vous au topic « foo/msg » sur le broker de test « test.mosquitto.org » et restez en attente :

```
//Abonnement au topic
mosquitto_sub -h test.mosquitto.org -q 1 -d -t foo/msg -i rapiSubscribe
```

Dans la deuxième fenêtre publiez un message sur le même topic et le même serveur :

```
//Publication du message
mosquitto_pub -h test.mosquitto.org -q 1 -d -t foo/msg -i raspiPublish -m "{\"msg\" :
 \"Hello, World\"}"
```

Vous devriez recevoir le message suivant sur la première fenêtre :

```
//Reception du message
{"msg" : "Hello, World"}
```

Maintenant que nous avons testé notre configuration nous avons besoin d'envoyer ce message vers le Cloud afin qu'il soit exploité par notre application. Le service le plus adapté pour faire cela est Amazon Kinesis, car c'est un service managé d'ingestion des données.
En plus d'être hautement disponible, ce service est également extrêmement élastique ce qui lui permet d'être adapté à une utilisation réelle et à très grande échelle.
Néanmoins la communication avec Amazon Kinesis est établie grâce au protocole HTTP(s), nous allons donc nous servir d'un pont pour recevoir les messages MQTT et les pousser en HTTPS vers Kinesis.

Etape 2 : Construction de la liaison entre l'objet connecté et Kinesis grâce à un bridge MQTT

Un bridge MQTT – Kinesis est disponible à l'adresse suivante :
https://github.com/awslabs/mqtt-kinesis-bridge
Vous pouvez suivre les instructions disponibles sur le fichier Readme.md pour déployer le bridge sur une instance Amazon EC2. Lors du lancement de votre instance EC2, pensez à associer un rôle IAM qui vous permettra de poster des messages dans Amazon Kinesis, pour cela associez une police IAM à votre rôle qui permettra à minima les actions suivantes : CreateStream, DescribeStream, GetRecords, GetShardIterator, ListStreams & PutRecord.
Pour l'instant nous allons uniquement tester l'installation de votre broker MQTT, vous pouvez le lancer à l'aide de la commande :

```
//Lancement du broker mqtt
mosquitto
```

Répétez ensuite les étapes mentionnées précédemment en remplaçant cette fois-ci le host par l'adresse IP publique de votre instance EC2, par exemple pour poster un message :

```
//Publication du message
mosquitto_pub -h XX.XX.XX.XX -q 1 -d -t foo/msg -i raspiPublish -m "{\"msg\" : \"Hello, World\"}"
```

Passons maintenant à l'étape de configuration de Kinesis avant de lancer le bridge MQTT à l'aide de Python.

Etape 3 : Ingestion des messages à l'échelle avec Kinesis

Kinesis peut collecter et stocker en permanence plusieurs téraoctets de données par heure à partir de centaines de milliers de sources, il est donc très adapté pour ingérer le flux d'informations provenant de nos appareils connectés.
La première étape consiste à créer un « Stream » Kinesis qui vous permettra de collecter tous les messages provenant de votre objet connecté. En réalité les messages provenant de l'objet connecté arriveront d'une instance EC2 qui abrite un pont MQTT vers Kinesis.
Pour cela rendez-vous sur la console AWS du service Kinesis et lancez la création d'un nouveau Stream, renseignez « bridgemqtt » pour le nom du stream et « 1 » pour le nombre de Shards (Fig.2).
Les Shards représentent l'unité d'élasticité de votre flux Kinesis; dans cet article nous n'avons besoin que d'un seul Shard, ce qui nous assure déjà de pouvoir ingérer 1 Mo/s de données. Dans un usage réel, il faudra prévoir l'utilisation de plusieurs Shards.

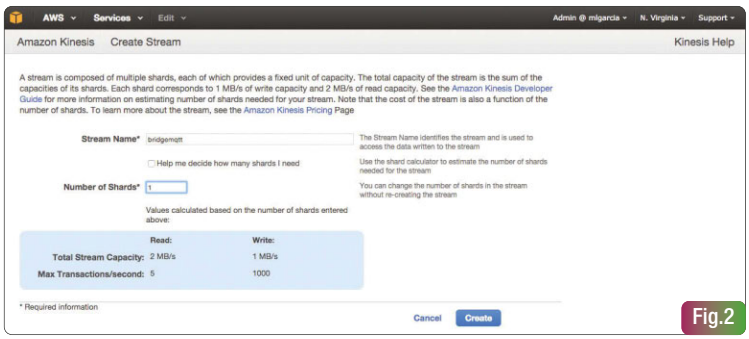


Fig.2

Etape 4 : Exploitez les informations collectées avec AWS Lambda

Il ne nous reste donc plus qu'à exploiter les informations venant du flux Kinesis : pour ce faire nous allons utiliser AWS Lambda.
Ce service permet d'exécuter du code en réponse à des évènements et gère automatiquement les ressources de calcul; dans notre cas précis les évènements sont les messages envoyés par notre objet connecté.
Dirigez-vous vers la console AWS Lambda pour créer votre première fonction Lambda en Node.js et sélectionnez le template « kinesis-process-record ».
Vient ensuite la configuration de la source d'évènements que notre fonction AWS Lambda exploitera. Pour cela sélectionnez « Kinesis » en tant que type de source d'évènements, « bridgemqtt » pour le nom du stream, « 100 » pour le Batch Size ce qui vous permettra de recevoir des évènements par paquet de 100 et « Latest » en tant que Starting Position, ce qui aura pour effet d'envoyer à AWS Lambda les derniers messages présents dans le flux Kinesis (Fig.3).
Dans l'écran suivant, donnez le nom « processMessages » à votre fonction lambda, et laissez les paramètres par défaut à l'exception du rôle de votre fonction. Choisissez ici « Kinesis execution role » pour créer un nouveau rôle (Fig.4) qui donnera les permissions nécessaires à votre fonction Lambda pour récupérer les messages contenus dans le flux Kinesis. Suivez ensuite les instructions à l'écran.
Pour la dernière étape de configuration choisissez d'activer la source d'évènements immédiatement.
Nous allons maintenant modifier ce template afin de pouvoir envoyer un message à notre objet connecté. Pour cela installez Node.js (https://nodejs.org/) si cela n'est pas déjà fait et exécutez la commande suivante sur votre terminal :

```
//Creation d'un nouveau dossier et installation du module MQTT pour Node.js
mkdir lambda && cd lambda && npm install mqtt
//Creation et édition de la function lambda
nano index.js
```

A l'aide de l'éditeur de fichier nano, copier/coller la fonction suivante en changeant l'adresse IP ci-dessous par celle de votre instance EC2 bridge MQTT:

```
//Initialisation du service Amazon Mobile Analytics
//Librairie MQTT
var mqtt    = require('mqtt');

exports.handler = function(event, context) {
    event.Records.forEach(function(record) {
        // Kinesis data is base64 encoded so decode here
        var payload = new Buffer(record.kinesis.data, 'base64').toString('ascii');
        console.log('Decoded payload:', payload);
    });
```

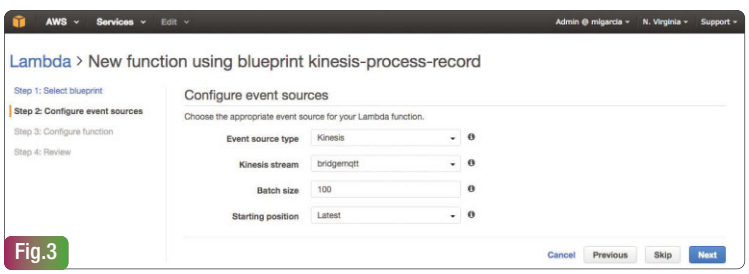


Fig.3

```
  //Connection au broker MQTT
  //Remplacez l'adresse IP par celle de votre instance bridge MQTT
  var client  = mqtt.connect('mqtt://XX.XX.XX.XX');

  //Publication du message vers l'objet connecté
  client.on('connect', function () {
   client.publish('foo/msg', "Successfully processed " + event.Records.length + " records.");
   client.end();
   context.succeed("Successfully processed " + event.Records.length + " records.");
  });
};
```

Il ne vous reste plus qu'à créer une archive du contenu de votre dossier grâce à la commande suivante, et à l'uploader manuellement via la console Web AWS lambda.

```
//Creation de l'archive avec le contenu du repertoire
zip –r lambda.zip .
```

Test de l'ensemble de la chaîne

Maintenant que tous les composants de la chaîne sont en place nous pouvons tester l'ensemble. Pour cela suivez les étapes ci-dessous en remplaçant l'adresse IP avec celle de votre instance EC2 bridge mqtt.
Lancez le broker mqtt sur votre instance EC2 :

```
//Lancement du broker MQTT
mosquitto
```

Lancez le bridge mqtt sur votre instance EC2 :

```
//Lancement du bridge MQTT Amazon Kinesis
python bridge.py bridgemqtt
```

Abonnez-vous au topic MQTT sur lequel la fonction lambda publiera un message depuis votre objet connecté :

```
//Abonnement au topic MQTT foo/msg, renseignez l'adresse IP de votre instance EC2
mosquitto_sub -h XX.XX.XX.XX -q 1 -d -t foo/msg -i rapiSubscribe
```

Publiez un message sur le topic du bridge mqtt depuis votre objet connecté :

```
//Publication sur le topic MQTT mqttkb/msg, renseignez l'adresse IP de votre instance EC2
mosquitto_pub -h XX.XX.XX.XX -q 1 -d -t mqttkb/msg -i raspiPublish -m "{\"msg\" : \"Hello, World\"}"
```

Vous devriez voir apparaître le message suivant sur votre objet connecté, ce message est publié par la fonction lambda et vous confirme que le traitement s'est bien effectué correctement :

```
//Réception du message venant de la fonction lambda
Successfully processed 1 records.
```

Mise en pratique avec une maison connectée

Le champ des possibilités est maintenant complètement ouvert. Vous êtes ainsi libre d'intégrer le code que vous souhaitez dans votre fonction Lambda. Nous avons envoyé à notre objet connecté un message de confirmation du traitement, ce même mécanisme vous permet d'envoyer des ordres à votre objet connecté pour le contrôler à distance.
De plus vous pouvez interagir avec d'autres services AWS pour créer tout type d'application afin de pouvoir effectuer du machine learning, de l'analytique et encore bien d'autres opérations.
Un premier exercice est de créer une application pour maison connectée. En connectant des LEDs, un buzzer et un capteur de température au Raspberry Pi nous pouvons émuler respectivement une lampe, une alarme et un thermostat.
Il sera alors possible de consulter en temps réel l'état de la lampe ainsi que la température dans une application Web serverless supportée par Amazon DynamoDB. Une application mobile pourra permettre au propriétaire de la maison de piloter sa lampe depuis l'extérieur de sa maison tout en recevant également les informations du système en temps réel. Enfin en cas d'incendie, la température relevée par le capteur dépassera un certain seuil toléré, à la suite de quoi le propriétaire de la maison sera alerté par une alarme (buzzer) et un appel téléphonique émis via une API tierce de service VOIP (Fig.5)

Conclusion

Vous venez de faire vos premiers pas dans le monde des objets connectés. Maintenant, vous pouvez compléter l'architecture présentée avec d'autres services de la plateforme AWS pour vous permettre de stocker vos données, de les analyser, de les rendre disponibles par API ou encore de vous intégrer avec votre propre système ainsi qu'avec des tiers.
Vous pouvez également interagir avec le monde réel maintenant que vous êtes capables d'envoyer des commandes à vos objets connectés. Cela vous offre de nouvelles opportunités d'innover quel que soit votre secteur d'activité grâce à ce nouveau type d'application.

Pour en apprendre davantage et démarrer votre premier projet avec des objets connectés rendez-vous dès à présent sur https://aws.amazon.com/iot/.

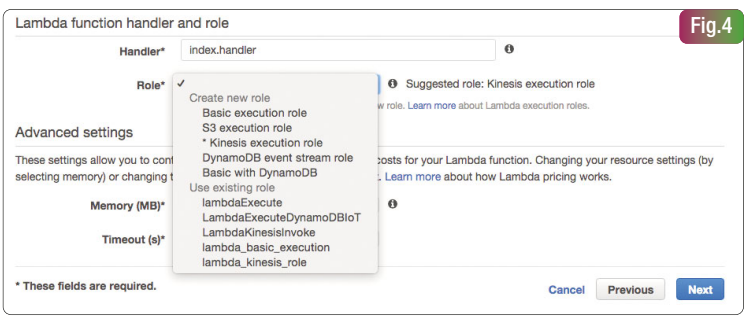


Fig.4

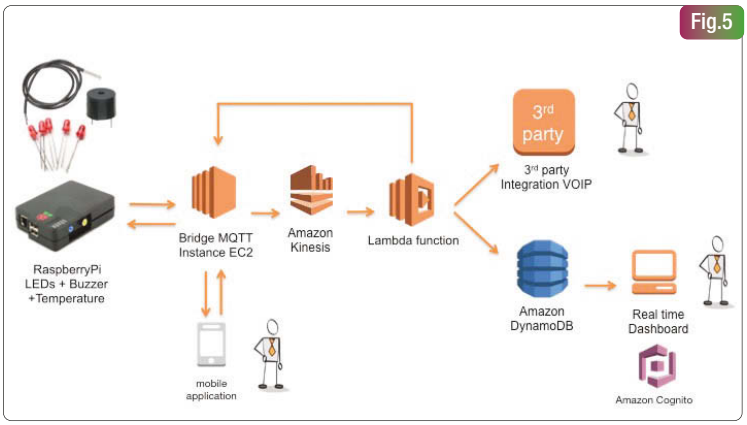


Fig.5

Insight : le contrôle qualité automatisé de vos projets PHP

Développeurs débutants ou initiés, architectes ou encore directeurs techniques, nous sommes souvent à la recherche de « guidelines », « best practices » permettant de concevoir des applications robustes et fiables dans le temps respectant l'état de l'art et facilement maintenables.

Antoine PACAUD
Directeur technique,
WEBNET

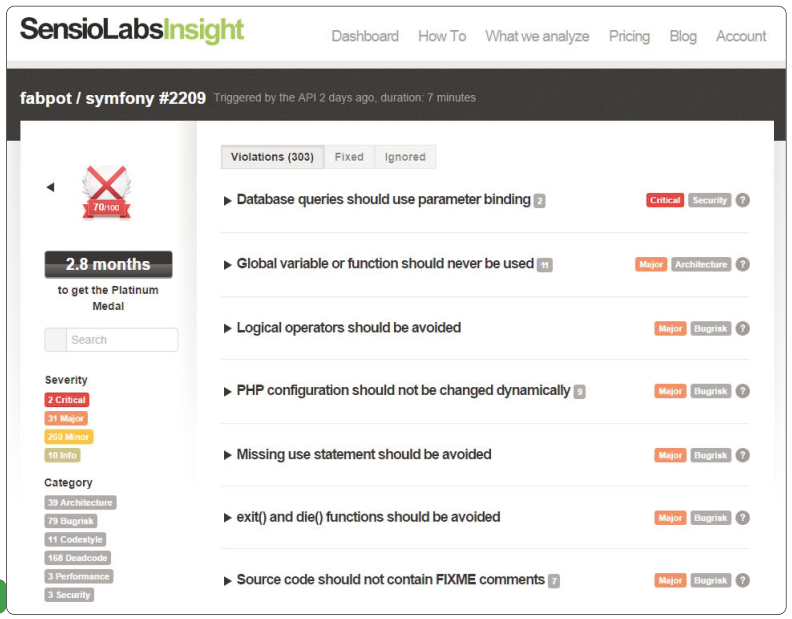


Fig.A

C'est dans cet esprit que Sensio, éditeur du framework Symfony2, a mis à disposition un outil nommé « Insight » permettant de contrôler et d'améliorer la qualité de projets utilisant le framework Symfony2, mais aussi d'autres frameworks PHP comme Silex, Laravel ou encore des modules Drupal.

Présenté en bêta lors des Symfony Live 2013 à Paris, il est désormais disponible dans sa version stable et commerciale et fonctionne en mode SaaS.

Il est uniquement disponible en ligne et ne nécessite aucune installation.

Après l'avoir testé durant plusieurs mois, c'est donc l'occasion pour moi de vous le présenter.

Plus d'une centaine d'indicateurs

Lorsqu'on soumet un projet Symfony2 au moteur d'Insight, celui-ci va tenter de l'installer grâce à Composer, puis de le lancer et d'y appliquer plus d'une centaine de tests divers et variés couvrant plusieurs domaines. L'une des forces du moteur est qu'il n'effectue pas qu'une analyse statique du code mais également une analyse dynamique. Celle-ci exécute l'application comme lorsqu'une requête arrive sur l'application. Le moteur est également capable d'analyser des fichiers non PHP tels que des templates TWIG ou encore des fichiers de configuration YAML ou XML. Voici quelques exemples des domaines couverts :

- La sécurité :
  - Utilisation de méthodes jugées peu fiables où présentant un risque de sécurité
  - Détection de failles SQL / XSS
  - Détection de la publication de données sensibles dans les sources
  - Traces de debug oubliées dans les sources
- Risque de bug
  - Détection d'erreurs de syntaxe ;
  - Détection d'une erreur d'installation / de compilation ;
  - Utilisation de méthodes dépréciées ;
  - Utilisation d'anciennes versions de framework comportant des bugs et/ou non maintenues.
- Performance
  - Complexité trop importante de méthodes ou de portions de code ;
  - Configuration non adaptée à la production.
- Architecture
  - Non-respect des bonnes pratiques ;
  - Utilisation de structures non optimisées ;
  - Duplication de code.
- Code mort
  - Code non utilisé ;
  - Présence de code commenté.
- Lisibilité du code / Respect des normes et « bonnes pratiques »
  - Respect des normes de développement (principalement PSR-2) ;
  - Longueurs des méthodes / classes.

En ce qui concerne la sécurité, Insight analyse les versions des librairies/bundles externes utilisées par votre projet (au moyen du fichier composer.lock) et les compare à sa base de données de failles de sécurité connues. Cela vous permet d'être alerté très rapidement si jamais une des librairies compromet la sécurité de votre application, vous délestant ainsi d'une partie de votre travail de veille.

Chaque anomalie détectée est ensuite pondérée par son degré de gravité (critique, majeure, mineure ou informative). Il en résulte une note globale sur 100 de la qualité de votre projet ainsi qu'une médaille la reflétant (de bronze, d'argent, d'or ou de platine si respectivement aucune anomalie critique, majeure, mineure ou informative n'est détectée).

Pour vous aider à la planification des corrections, chaque anomalie est également accompagnée d'une estimation du temps nécessaire à sa correction. Chaque résultat d'analyse est présenté sous forme d'un tableau de bord représentant l'ensemble de ses informations (fig. A). Il est possible de consulter le détail de chacune des anomalies afin d'obtenir les fichiers concernés, l'estimation du temps nécessaire à la correction, ainsi que la marche à suivre (fig B).

On pourra regretter la faible pertinence de l'estimation temporelle des corrections. En effet, le temps minimal de correction d'une anomalie est estimé à 15min. Si cela peut s'expliquer par le temps nécessaire à ouvrir le projet, réaliser la correction, déployer ses modifications, etc, cette estimation perd en revanche, de son sens lorsque 10 anomalies du même type sont remontées. En effet, l'outil va estimer qu'il faudra 1h30 (150min) pour réaliser la correction alors qu'en pratique 20min suffiront dans la plupart des cas. Il s'agit donc d'un indicateur à prendre avec précaution et qui devra être tempéré par l'équipe technique.

Implémentation au sein d'une solution d'intégration continue

Pour profiter pleinement de cet outil et maîtriser la qualité de votre projet, il est souhaitable que ces tests soient effectués très régulièrement, idéalement chaque jour, ou même après chaque modification du code source.

Pour commencer, Insight va avoir besoin d'accé-

der au code source de votre application. Plusieurs moyens sont proposés pour cela. Si votre code source est hébergé sur des plateformes telles que GitHub ou Bitbucket, il vous est possible de donner la permission à l'outil d'y accéder directement afin de pouvoir y exécuter ses différents tests. Si vous hébergez vous-même votre propre dépôt GIT, vous pouvez également y donner l'accès à Insight sous réserve de l'ouvrir à ses serveurs au moyen d'une protection par clés SSH. Enfin, si vous ne souhaitez pas ouvrir l'accès à votre dépôt ou que vous utilisez un autre système de versionning tel que SVN, Insight propose de vous créer un dépôt GIT sur leur plateforme sur lequel envoyer vos sources quand vous souhaitez les faire analyser. Une option vous permet par ailleurs de demander à l'outil de détruire la copie de vos sources après chaque analyse pour des problématiques de confidentialité.

Une fois l'accès accordé à l'outil, il va vous être possible de programmer les analyses de différentes manières. Dans le cas de GitHub ou de Bitbucket, vous allez pouvoir exploiter la fonctionnalité de « hook », et déclencher ainsi automatiquement les analyses après chaque « push ». Une fois l'analyse effectuée, il vous sera également possible d'ouvrir des tickets automatiquement via l'interface et ainsi d'affecter les corrections à vos collaborateurs.

Insight prend également sa place tout naturellement au sein d'une solution d'intégration continue telle que Jenkins, Hudson, Travis, pour ne citer qu'eux. Il propose ainsi un petit utilitaire en ligne de commande qui va permettre à ces outils de piloter les analyses et d'en récupérer les résultats. Il devient alors possible d'afficher directement les résultats de l'analyse au sein du dashboard de Jenkins par exemple.

Et en pratique ?

Après une première analyse nécessitant quelques minutes, le verdict et la première note tombe. Autant vous prévenir, il est rare d'obtenir mieux que la médaille de bronze dès le premier essai ! C'est là où le côté "gamification" de l'outil prend le pas (au moyen des médailles et notes), et on se surprend vite à vouloir obtenir le niveau supérieur quitte à faire quelques heures supplémentaires !

On prend vite le pas et on apprécie que l'outil nous rappelle nos erreurs d'inattention comme la longueur (20 lignes maximum) et le nombre de nos actions (10 max/contrôleur), le respect des normes PSR-2 ou simplement le bon découpage de la configuration de notre projet.

Il en résulte un code plus maintenable, plus élégant, sur lequel il est plaisant de travailler.

L'outil permet également à la personne en charge de la revue qualité de se concentrer sur l'algorithmique, l'architecture générale et le métier, plus que sur ces aspects qui sont désormais audités automatiquement.

Le tableau de bord est bien conçu et le détail de chaque anomalie avec l'explication liée, et les pistes de résolution sont plutôt efficaces. Si, au début, l'outil remontait régulièrement des faux positifs, l'équipe Sensio semble avoir pris en compte les remontées des utilisateurs et, aujourd'hui, la pertinence des alertes est bonne. Il reste d'ailleurs toujours possible d'ignorer une alerte particulière (ou même un groupe d'alertes), si celle-ci vous semble inexacte ou inadaptée à votre projet.

Quel niveau viser ?

Il est légitime de se poser la question du niveau de qualité minimal à atteindre avec ce genre d'outil.

Il n'y a pas de règle absolue, cela dépend de l'ancienneté de votre projet, de l'équipe qui y contribue et de leur niveau de compétence sur la technologie utilisée, mais une chose est sûre : plus haute sera la note, meilleur sera votre projet. On pourrait résumer les médailles comme ceci :

- Pas de médaille : votre projet comporte, au minimum, un risque de sécurité majeur et/ou un problème de syntaxe dans un des fichiers. Vous devriez corriger rapidement les anomalies remontées.
- Médaille de bronze : La plupart des anomalies remontées à ce niveau révèlent des défauts importants de votre application (utilisation de fonctions PHP dangereuses et mauvaise architecture de votre application dans la plupart des cas). Votre application gagnerait beaucoup en maintenabilité si vous passiez ce cap.
- Médaille d'argent : Les principales « best-practices » sont respectées par votre projet. C'est le niveau minimal pour un projet en cours de développement selon moi. En revanche, les alertes remontées restent importantes et peuvent traduire un manque de rigueur dans le développement. La plupart d'entre elles ne nécessitent pas un investissement lourd pour être corrigées alors pourquoi s'en priver ?
- Médaille d'or : Bravo ! Votre projet respecte la grande majorité des « best-practices » et son architecture semble très correcte. Ce niveau peut tout à fait être suffisant pour un projet en cours de développement mais après tout, pourquoi s'arrêter en si bon chemin ?
- Médaille de platine : Kudos! Pour reprendre les termes d'Insight. L'outil n'a pas été capable de trouver un seul point faible à votre projet. C'est la récompense ultime et le stade idéal d'une release en production de votre projet.

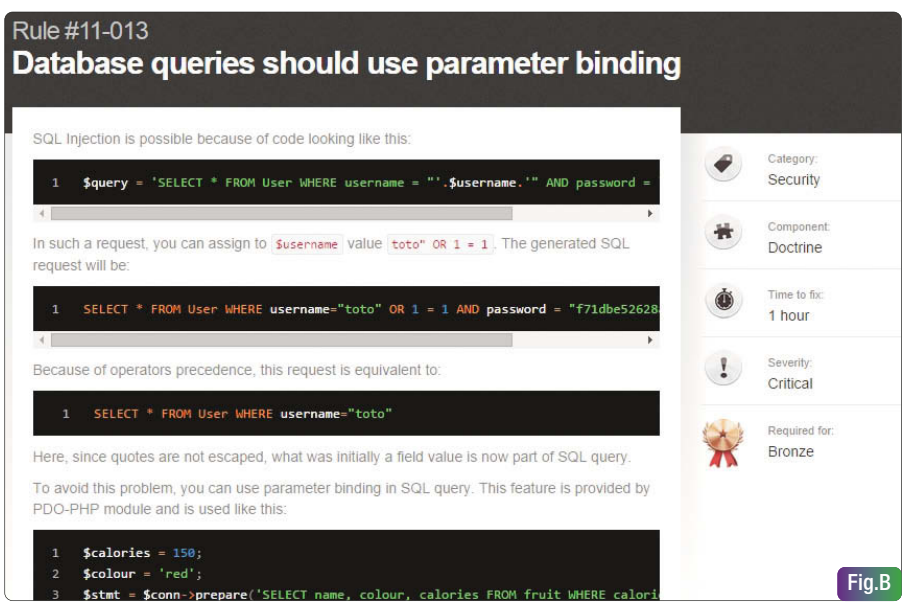


Fig.B

Conclusion

Sensio présente donc un outil performant qui vous aidera à améliorer rapidement et de façon certaine la qualité globale de vos projets. Cet outil devrait être idéalement utilisé dès le début du chantier afin de pouvoir traiter au fur et à mesure les remontées. En effet, sur un gros projet déjà développé, la charge de travail de correction des alertes remontées pourrait être vite trop lourde et décourageante.

Bien que stable, l'outil continue d'évoluer chaque mois via l'ajout de nouvelles règles et de nouvelles fonctionnalités. Il est proposé gratuitement pour les projets open-source ou via des plans payants (à partir de 6€/mois) pour les particuliers ou professionnels.

Pour ma part, l'essayer fut l'adopter et l'objectif fixé est la médaille de platine à chaque fin de projet !

Développer avec le SDK Kinect 2

1ère partie

La Kinect 2 est le capteur de mouvements de deuxième génération produit par Microsoft. La Kinect intègre plusieurs capteurs dont une caméra couleur full HD, un émetteur et un récepteur infrarouge ainsi que quatre micros. De par la diversité et la nature de ses composants, exploités par le SDK de développement Kinect 2 pour Windows, la Kinect offre au développeur toutes sortes de possibilités d'interaction avec l'humain très facilement exploitables par le SDK.

Thomas Ouvré
Consultant Infinite Square
Blog : http://blogs.infinitesquare.com/b/touvre
INFINITE SQUARE

En développant une application pour Kinect 2, le panel de plateformes cibles est très large. Cela peut aller d'une application pour Xbox à une application Desktop Windows 8 en passant par les applications Windows Store, ou Windows Embedded. Les plateformes étant variées, il en va de même pour les langages et technologies. N'importe quel langage couramment utilisé dans l'environnement Microsoft, à savoir C++, C# ou JavaScript (pour les applications WinRT). Il est même possible d'utiliser le SDK au sein d'une application Unity.

Le SDK est téléchargeable à l'adresse suivante : https://www.microsoft.com/en-us/kinectforwindows/develop/. Pour fonctionner, il est recommandé de disposer d'une machine avec au moins 4 go de RAM et une carte graphique compatible DirectX 11. Il est aussi nécessaire de disposer d'un contrôleur USB 3 dont la bande passante soit suffisante pour la quantité de données transitant entre la Kinect et la machine. Une fois le SDK installé, il est possible d'utiliser l'outil Kinect Configuration Verifier accessible via l'application SDK Browser v2.0 (Kinect for Windows) pour s'assurer que la configuration de la machine actuelle est suffisante : Fig.1.

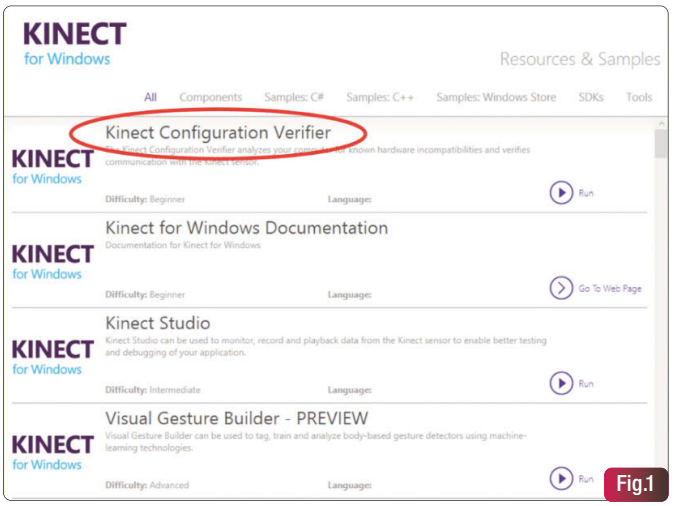


Fig.1

SDK BROWSER ET OUTILS OU COMMENT DÉVELOPPER POUR KINECT SANS KINECT

Outre l'outil Kinect Configuration Verifier, l'installation du SDK offre l'accès à plusieurs outils très pratiques pour le développeur Kinect. Il est en effet intéressant de prendre connaissance des capacités de ces outils avant de commencer à développer pour Kinect car ils sont en mesure de faciliter la vie du développeur, quitte même à permettre de développer sans Kinect physique pour l'un d'entre eux.

Kinect Studio

L'outil Kinect Studio a pour but premier de monitorer la Kinect. Ainsi, une fois lancé, dès la connexion établie entre la machine et le capteur, il permet de visualiser un rendu « possible » des différentes caméras. On peut y jouer avec les différentes options permettant de choisir l'une ou l'autre des caméras, ou encore sélectionner l'un des flux sources exposés par le SDK, et ce, de différentes manières. Voici un aperçu de l'outil et de l'onglet Monitor dont il est question : Fig.2.

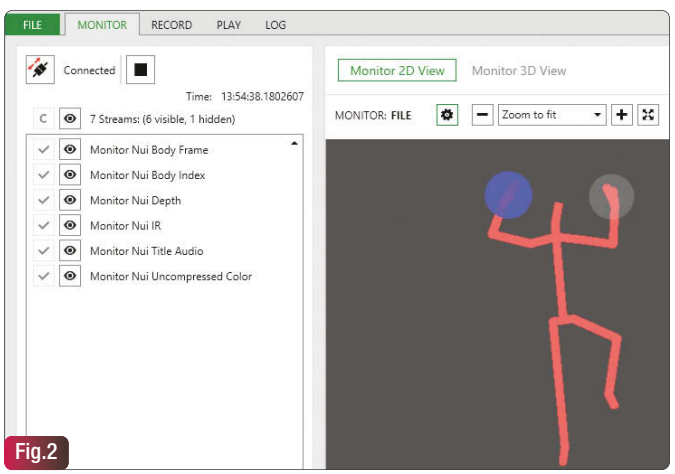


Fig.2

L'autre aspect intéressant de Kinect Studio est sa capacité à enregistrer les données transmises par le capteur. Cette opération se fait via l'onglet Record. Cette option va nous être utile car les fichiers issus de cet enregistrement (appelés Clips) vont nous permettre de rejouer l'ensemble de la scène enregistrée dans un Clip ; il sera ainsi possible d'analyser trame par trame une scène en utilisant la timeline et surtout de simuler la Kinect dans notre application. En effet, Kinect Studio peut agir comme une sorte de simulateur de la Kinect du point de vue du SDK et permettre de développer, debugger et tester une application utilisant le SDK Kinect sans être nécessairement relié à un capteur physique. Il s'agit donc d'une bonne nouvelle pour tout développeur souhaitant s'initier à ce SDK (en lisant la suite de cet article par exemple) sans pour autant disposer d'une Kinect 2. Des clips prêt à l'emploi sont d'ailleurs disponibles sur la plateforme Microsoft Virtual Academy en suivant ce lien (http://www.microsoftvirtualacademy.com/training-courses/introduction-au-developpement-kinect-v2, télécharger le code source de la première vidéo). D'autres sont aussi disponibles partout sur le Web.

L'autre intérêt des clips produits par Kinect Studio est qu'ils sont les fichiers d'entrée nécessaires à la création de projets pour Visual Gesture Builder.

Visual Gesture Builder

Avant de se lancer dans un développement très compliqué visant à effectuer de la reconnaissance de gestes, à traquer des postures et des gestures, tout développeur Kinect doit connaître l'existence de ce formidable outil qu'est Visual Gesture Builder. En effet, le traitement des données brutes accessibles via le SDK nécessite la mise en place d'algorithmes complexes, de formules mathématiques et autres. Or, ce n'est pas néces-

sairement le genre de challenge que recherchent les développeurs actuellement, moi le premier. Visual Gesture Builder permet au développeur de réduire drastiquement la complexité de mise en place de tels scénarios, et même d'en assurer une meilleure efficacité. En effet, plutôt que de mesurer des déplacements de points, de calculer des vitesses et d'effectuer des interpolations et autres algorithmes à base de fonctions mathématiques imbuvables, le développeur Kinect peut faire appel à la puissance du machine learning et créer des bases de données de gestures, laissant ainsi les APIs du SDK faire le travail de détection.

Dans un scénario d'usage avec Visual Gesture Builder, les clips de Kinect Studio servent à définir les moments pendant lesquels une gesture est réalisée par un utilisateur (le développeur doit taguer des scènes). Le moteur de machine learning est ensuite capable d'ingérer ces informations et de définir des patterns de détection de ces gestures. Bien entendu, plus le nombre de clips fournis est important et plus le travail de détection effectué par le développeur est précis et correct, plus les patterns générés seront précis et efficaces. Pour permettre à VGB d'apprendre à détecter une gesture, le développeur n'a qu'à apposer des tags sur chaque trame d'un clip pour définir si une gesture est effectuée à un instant t.

Bien entendu, ce scénario est volontairement simplifié car il n'est pas le sujet de cet article, mais en parler dès maintenant permettra peut-être à certains de s'orienter rapidement vers cette solution potentiellement bien plus simple et efficace à mettre en place que des « calculs maison ». Pour se donner une idée de ce que VGB permet de réaliser, un sample est livré avec l'installation du SDK.

Samples

Tous les samples sont accessibles via l'outil SDK Browser, dont les applications Discrete Gesture Basics – WPF et Discrete Gesture – XAML. Ces dernières sont un exemple d'application utilisant une base de données produite par VGB pour la détection d'une gesture discrète : la position assise. Ces exemples vont de l'usage des micros de la Kinect à la détection d'un squelette en passant par le rendu de la caméra infrarouge, de profondeur ou de couleur... La reconnaissance vocale, la reconstruction 3D, la détection d'expressions et l'application Kinect Evolution sont autant de sources d'informations disponibles pour l'apprentissage du SDK Kinect.

Néanmoins, la suite de cet article est aussi un point d'entrée dans l'apprentissage du SDK puisqu'elle vise à la création d'une application C# Windows Store manipulant les fonctionnalités de base du SDK.

APPLICATION C# / WINDOWS STORE

Référencer le SDK et autoriser la Kinect

Une application Windows Store peut référencer le SDK Kinect via l'interface de gestion des références. L'onglet Windows 8.1 > Extensions permet de lister tous les composants WinRT accessibles, dont WindowsPreview.Kinect : Fig.3.

Ce composant dépendant directement de Microsoft Visual C++ Runtime Package, ce dernier est normalement automatiquement ajouté à la liste des références.

Une chose à rappeler : la Kinect, une fois branchée, apparaît entres autres choses comme une webcam et un micro. Il ne faut donc pas oublier d'aller sur le manifeste du projet, couramment appelé Package.appxmanifest, et de cocher dans l'onglet Capabilities les options Microphone et Webcam.

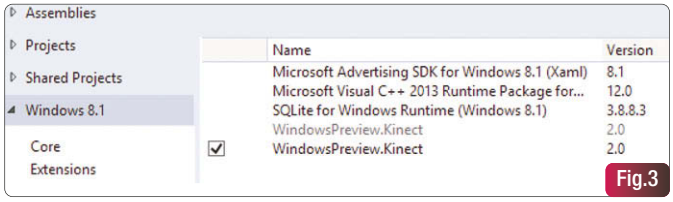


Fig.3

Toutes les classes nécessaires sont accessibles via le namespace WindowsPreview.Kinect. On y trouve notamment la classe KinectSensor correspondant au point d'entrée de l'API Kinect. Une instance de cette classe représente une connexion entre le capteur et la machine et permet d'accéder à différentes sources de données. Voici tout d'abord comment obtenir une telle instance :

```
var _sensor = KinectSensor.GetDefault();
if (!_sensor.IsOpen)
        _sensor.Open();
```

Cet objet permet de déterminer si le capteur est activé et de l'activer si ce n'est pas le cas grâce à la méthode Open si c'est possible, une connexion sera ouverte et des données pourront être reçues depuis la Kinect. De plus, l'appel de cette méthode permet d'établir automatiquement une connexion à un capteur lorsque celui-ci est physiquement disponible, même si cela se produit après cet appel. Le modèle de développement suggéré par l'API fait que le développeur n'a pas à se soucier de cela tant qu'il respecte certains patterns. Néanmoins, s'il souhaite s'assurer qu'un capteur physique est bien disponible, il peut faire appel à la propriété IsAvailable et à l'évènement IsAvailableChanged.

L'instance de KinectSensor expose ensuite plusieurs propriétés dont les noms se terminent par Source. Ces propriétés sont les points d'entrées vers des fonctionnalités précises de l'API que l'on appellera sources de données. Ces sources permettent entre autres d'accéder à des flux, comme celui des caméras de la Kinect, ou à des flux de plus haut niveau issus de calculs effectués sur les premiers. Ainsi, les flux bruts et leurs points d'entrée correspondants sont :

- Infrarouge à InfraredFrameSource
- De profondeur à DepthFrameSource
- De couleur à ColorFrameSource
- Infrarouge surexposé à LongExposureInfraredFrameSource
- Audio à AudioSource

D'autres flux sont directement issus de calculs effectués sur les premiers (et notamment celui de profondeur), ce qui nous évite pas mal d'efforts :

- Body index à BodyIndexFrameSource
- Body à BodyFrameSource

D'autres sont accessibles via des composants WinRT supplémentaires comme un flux spécialisé pour les scénarios d'usage de VGB, ou un autre pour le traitement du visage et des expressions : Face.

Quel que soit le flux que le développeur souhaite manipuler, il le fera en utilisant une méthode OpenReader sur la source. Celle-ci va permettre d'établir une connexion avec une source précise et de récupérer les dernières données (grâce à une méthode dont le nom commence par AcquireLatest) ou de s'abonner à un évènement (FrameArrived) et être notifié lorsque de nouvelles données sont disponibles. Ce pattern est commun à toutes les sources.

Les caméras

Caméra Couleur

Le premier flux le plus simple à afficher est celui de la caméra couleur. En effet, les trames que renvoie ce flux sont de simples bitmap formatés. Or les APIs Kinect permettent de définir un format de conversion dont le format BGRA. Ce format, extrêmement simple à manipuler, est directement compatible avec celui attendu d'un contrôle <Image>. Un moyen simple d'effectuer un rendu temps réel du flux de la caméra couleur consiste en la mise en place d'un contrôle Image dont la source est un WriteableBitmap. A chaque trame disponible, ce dernier sera modifié en conséquence afin

de mettre à jour l'image. Le taux de rafraîchissement de la caméra couleur étant au maximum de 30 fps la fluidité est optimale. De plus, les trames couleurs sont dans une résolution suffisante pour que la qualité soit aussi au rendez-vous : 1920x1080 !
Bien entendu, ces derniers chiffres sont nécessaires dans le code, car ils permettent d'initialiser le WriteableBitmap. Le développeur doit donc faire appel à la description de la trame dans laquelle se trouvent les informations nécessaires. Dans le cas du flux de la caméra couleur, une méthode permet d'obtenir la description des trames dans le format BGRA :

```
var source = _sensor.ColorFrameSource;
var description = source.CreateFrameDescription(ColorImageFormat.Bgra);
var width = description.Width;
var height = description.Height;
```

Il suffit ensuite d'initialiser un WriteableBitmap et l'assigner à un contrôle Image nommé myImage

```
_bitmap = new WriteableBitmap(width, height);
myImage.Source = _bitmap;
```

La suite consiste logiquement à acquérir le reader du flux courant, l'ouvrir et s'abonner à l'évènement FrameArrived comme suit :

```
_colorReader = source.OpenReader();
_colorReader.FrameArrived += Reader_FrameArrived;
```

Le handler Reader_FrameArrived associé à l'évènement FrameArrived (levé dès qu'une nouvelle trame est disponible et exploitable) sera utilisé pour la mise à jour du WriteableBitmap. Les arguments de l'évènement sont utilisés pour récupérer la trame et copier les données dans le format BGRA attendu par notre WriteableBitmap :

```
private void Reader_FrameArrived(ColorFrameReader sender, ColorFrameArrivedEventArgs args)
{
  using (var frame = args.FrameReference.AcquireFrame())
  {
    if (frame == null) return;
    frame.CopyConvertedFrameDataToBuffer(_bitmap.PixelBuffer, ColorImageFormat.Bgra);
    _bitmap.Invalidate();
  }
}
```

Reste à ajouter le contrôle Image dans le XAML si ce n'est déjà fait :

```
<Image x:Name="myImage" />
```

Et éventuellement une méthode permettant de stopper le flux courant et libérer les ressources allouées :

```
private void StopCurrentDemo()
{
  if (_colorReader != null)
  {
    _colorReader.FrameArrived -= Reader_FrameArrived;
_colorReader.Dispose();
_colorReader = null;
  }
  myImage.Source = null;
  _bitmap = null;
}
```

Que ce soit avec une Kinect branchée ou avec Kinect Studio en mode émulation, cet exemple permet d'afficher un rendu très fluide de la caméra couleur Fig.4.


Fig.4

Infrarouge et profondeur

Les flux infrarouges et de profondeur sont la base des capacités de reconnaissance de la Kinect et du SDK. Cette dernière va permettre de mesurer les distances se trouvant entre le capteur et les objets qui lui font face. La mesure de ces distances sur une scène permet aux APIs après divers calculs d'exposer d'autres sources de données de plus haut niveau. La caméra infrarouge quant à elle permet principalement la mise en place de scénario où l'analyse des textures est importante comme la reconnaissance faciale. L'utilisation de ces deux flux se fait de la même façon que la caméra couleur. Ainsi, si le but est d'afficher le rendu de la caméra infrarouge, on peut se contenter d'adapter l'exemple précédent. Néanmoins, les données recueillies, bien qu'étant aussi des tableaux de pixels pour chaque trame, ne peuvent pas être utilisées telles quelles dans un bitmap. En effet, une intensité du signal infrarouge n'a rien à voir avec une couleur BGRA par exemple. Il faut donc interpréter la donnée pour en faire une couleur, comme un niveau de gris.

```
var source = _sensor.InfraredFrameSource;
var description = source.FrameDescription;
var width = description.Width;
var height = description.Height;
_bitmap = new WriteableBitmap(width, height);:
_infraredBuffer = new ushort[description.LengthInPixels];
```

On stockera le pixel sous forme de byte au format BGRA (où la constante BGRA_BYTES_PER_PIXEL vaut 4) :

```
_bitmapIntermediateBuffer = new byte[width * height * BGRA_BYTES_PER_PIXEL];
```

Reste ensuite à définir le bitmap en tant que source de l'image et à s'abonner à l'évènement FrameArrived comme précédemment :

```
myImage.Source = _bitmap;
_infraredReader = source.OpenReader();
_infraredReader.FrameArrived += Reader_FrameArrived;
```

Comme précédemment nous récupérons les informations depuis les arguments de la fonction. Ce qui change cette fois-ci est la recopie des données ; cette fois, il n'y a pas de transformation que l'API peut effectuer pour créer un tableau de pixels compatible avec le bitmap. Il faut donc obtenir ce tableau sous un format brut avant d'effectuer une transformation manuellement :

```
private void Reader_FrameArrived(InfraredFrameReader sender, InfraredFrameArrivedEvent
Args args)
{
  using (var frame = args.FrameReference.AcquireFrame())
  {
    if (frame == null) return;
    frame.CopyFrameDataToArray(_infraredBuffer);
    //suite ici...
    _bitmap.Invalidate();
  }
}
```

Chaque valeur du tableau _infraredBuffer correspondant à une intensité du signal infrarouge, que nous convertissons dans une valeur située entre 0 et 255 (minimum et maximum d'un byte) permet de remplir le tableau _bitmapIntermediateBuffer :

```
for (var i = 0; i < _infraredBuffer.Length; ++i)
{
  var intensity = _infraredBuffer[i];
  const float CONTRAST = 1.2F;
  var color = (byte)((intensity + CONTRAST * intensity) * byte.MaxValue / short.MaxValue);
  var j = i * BGRA_BYTES_PER_PIXEL;
  _bitmapIntermediateBuffer[j + 0] = color; //blue
  _bitmapIntermediateBuffer[j + 1] = color; //green
  _bitmapIntermediateBuffer[j + 2] = color; //red
  _bitmapIntermediateBuffer[j + 3] = byte.MaxValue; //alpha
}
```

Dans cet exemple, la constante CONTRAST permet d'influer sur le niveau de gris obtenu : plus la valeur est élevée, plus l'image obtenue est contrastée et claire. En effet, si l'on garde cette valeur à 0, on se rend compte que l'image produite est très sombre. La constante BGRA_BYTES_PER_PIXEL représente le nombre d'octets nécessaires à l'expression d'une couleur sous le format BGRA. Elle permet de se placer correctement dans le tableau _bitmapIntermediateBuffer qui sera compatible avec notre WritableBitmap.
Nous allons donc copier ce bitmapIntermediateBuffer dans l'objet _bitmap qui sera la source de notre contrôle image.

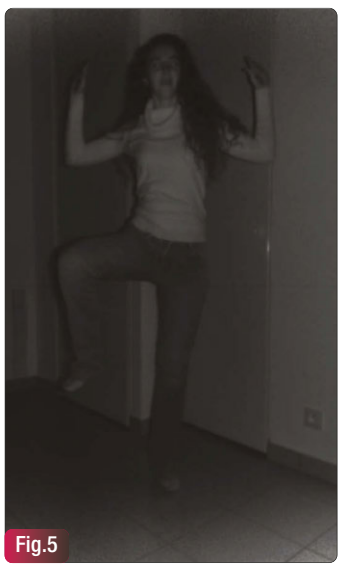


Fig.5

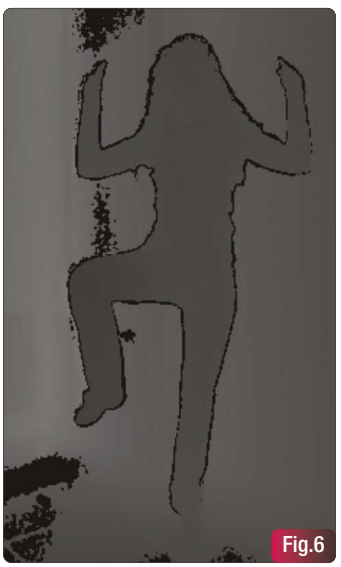


Fig.6

```
_bitmapIntermediateBuffer.CopyTo(_bitmap.PixelBuffer);
```

Dans un scénario complet, on n'oubliera pas de compléter la méthode StopCurrentDemo afin de permettre le relâchement des ressources impliquées. Voici un aperçu du rendu produit par ce code : Fig.5.
Dans le cas du flux de la caméra de profondeur, l'utilisation de l'API correspondante est d'autant plus similaire que les données de la trame de profondeur sont une valeur correspondant à la distance se trouvant entre l'objet et le capteur, exprimée en millimètres.
Finalement, pour en produire une représentation visuelle, l'exemple précédent peut entièrement être repris à ceci près qu'il faut y remplacer le terme infrared par depth. On aura donc à l'initialisation :

```
var source = _sensor.DepthFrameSource;
var description = source.FrameDescription;
var width = description.Width;
var height = description.Height;
_bitmap = new WriteableBitmap(width, height);
_bitmapIntermediateBuffer = new byte[width * height * BGRA_BYTES_PER_PIXEL];
_depthBuffer = new ushort[description.LengthInPixels];
myImage.Source = _bitmap;
_depthReader = source.OpenReader();
_depthReader.FrameArrived += Reader_FrameArrived;
```

Puis le handler effectuant la copie :

```
private void Reader_FrameArrived(DepthFrameReader sender, DepthFrameArrivedEvent
Args args)
{
  using (var frame = args.FrameReference.AcquireFrame())
  {
    if (frame == null) return;
    frame.CopyFrameDataToArray(_depthBuffer);
    for (var i = 0; i < _depthBuffer.Length; ++i)
    {
      var depth = _depthBuffer[i];
      var color = (byte)(depth * byte.MaxValue / 8000);
      var j = i * BGRA_BYTES_PER_PIXEL;
      _bitmapIntermediateBuffer[j + 0] = color; // blue
      _bitmapIntermediateBuffer[j + 1] = color; // green
      _bitmapIntermediateBuffer[j + 2] = color; // red
      _bitmapIntermediateBuffer[j + 3] = byte.MaxValue; //alpha
    }
    _bitmapIntermediateBuffer.CopyTo(_bitmap.PixelBuffer);
    _bitmap.Invalidate();
  }
}
```

La seule différence se trouve au niveau de l'instanciation de la variable color. Dans ce cas précis, il n'est pas utile d'influer sur le contraste, mais de modifier la valeur maximum soit 8000 (correspondant à 8 mètres). Ainsi l'image est bien plus contrastée : Fig.6. ⊞

Suite et fin dans le prochain numéro.

Comment gagner du temps de dév très facilement ?

Abonnement : Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com* - Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30. **Tarifs** abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € Autres pays : nous consulter.
PDF : 30 € (Monde Entier) souscription sur www.programmez.com

Une publication **Nefer-IT**
7 avenue Roger Chambonnet
91220 Brétigny sur Orge
redaction@programmez.com
Tél. : 01 60 85 39 96

Directeur de la publication & rédacteur en chef : François Tonic

Secrétaire de rédaction : Olivier Pavie

Ont collaboré à ce numéro : S. Saurel

Experts : V. Quadrelli, C. Villeneuve, M. Garcia, A. Pacaud, T. Ouvré, H. Carnicelli, R. Foucaud, T. Templier, M. Fery, S. Cordonnier, A. Talavera, T. Lebru, C. Pichaud, K. Sibué, F. Fadel, P. Batty, L. Claustres, C. Peugnet, J. Doucet, R. Niveau, Bouhanef Hamdi, Dridi Manel, A. Benabdi, F. Noël, W. Chegham.

Photos/illustrations : © maxsattana

Maquette : Pierre Sandré

Publicité : PC Presse,
Tél.: 01 74 70 16 30, Fax : 01 41 38 29 75
pub@programmez.com

Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.

Marketing et promotion des ventes :
Agence BOCONSEIL - Analyse Media Etude

Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD
Téléphone : 09 67 32 09 34

Contacts
Rédacteur en chef : ftonic@programmez.com
Rédaction : redaction@programmez.com
Webmaster : webmaster@programmez.com
Publicité : pub@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1215 K 78366 - ISSN : 1627-0908

Sur abonnement ou en kiosque

Le magazine des pros de l'IT

Mais aussi sur le web

Ou encore sur votre tablette

LE CLOUD GAULOIS, UNE RÉALITÉ ! VENEZ TESTER SA PUISSANCE

EXPRESS HOSTING

Cloud Public
Serveur Virtuel
Serveur Dédié
Nom de domaine
Hébergement Web

sales@ikoula.com
01 84 01 02 66
express.ikoula.com

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

sales-ies@ikoula.com
01 78 76 35 58
ies.ikoula.com

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

sales@ex10.biz
01 84 01 02 53
www.ex10.biz