PHP 5.3 | REACT | GITLAB | POWERSHELL | LINUX

programmez.com

Programmez!

Le magazine des développeurs

222 OCTOBRE 2018

LES DÉVELOPPEURS PEUVENT-ILS VIVRE DE LEURS APPS ?

Avoir la bonne idée ne suffit pas !

© RapidEye

OpenStreetMap

L'alternative à Google Maps

Bazic
Créer son propre langage de développement

CLego
Un cluster en Lego !

L'IA veut-elle tuer le développeur ?

M 04319 - 222 - F: 6,50 € - RD

LE SEUL MAGAZINE ÉCRIT PAR ET POUR LES DEVELOPPEURS

Printed in EU - Imprimé en UE - BELGIQUE 7 e - Canada 9,80 $ CAN - SUISSE 13,10 FS - DOM Surf 7,50 e - TOM 1020 XPF - MAROC 55 DH

DEVOPS D-DAY #4

2018

Jeudi 15 novembre

Orange vélodrome, Marseille

La conférence IT la plus attendue en France

40 conférences

Agilité - Cloud - Conteneurisation - DevOps - Kubernetes
Méthodologie - Microservices - REX - Serverless - UX
Transformation Digitale - Libre

libday
Conférences
Logiciel Libre

Sponsors Platinum

TREEPTIK.
A LINKBYNET COMPANY

NUTANIX
YOUR ENTERPRISE CLOUD

drops

vmware
axians

www.devops-dday.com
2018.libday.fr

EDITO

Tu utilises mes API, pas grave, je te taxe de 500 % !

Il y a des jours où on se demande « mais pourquoi ils font des trucs comme ça ? » ou encore « parfait pour faire fuir les développeurs ». Bref, certains éditeurs / services plombent la communauté, pour de bonnes ou de mauvaises raisons.

C'est un jeu de « je t'aime moi non plus ». Mais il y a tout de même des fondamentaux : le succès d'une plateforme passe aussi par les développeurs. Et aujourd'hui, plus que dans les années 80 et 90, le développeur est au centre de tout. Un app store va réussir si les apps le remplissent.

Jusqu'au début des années 2010 (oui je sais c'est déjà la préhistoire), c'était un peu l'inverse. Les ventes d'un device incitaient les éditeurs et développeurs à y aller.

Les exemples d'API, de SDK, de services arrêtés sans ménagements sont nombreux. Et il y a les éditeurs qui modifient les tarifs et les conditions d'usage des API et services… Et là, ça peut faire mal, très mal. L'exemple le plus récent est celui de Google avec le service Maps : tarifs en forte hausse, limitation des quotas gratuits. Cette décision a incité des développeurs à regarder les alternatives, comme nous le verrons dans notre dossier consacré à OpenStreetMap.

Ne pas avoir de visions claires sur une plateforme n'aide pas non plus. Microsoft avec Windows Mobile est sans doute l'exemple le plus représentatif de ces dernières années. Plus récemment, c'est Twitter qui coupe certaines API pour favoriser ses propres apps…

Les scandales à répétition sur les données personnelles ou encore les accès silencieux à certaines fonctions d'un device ternissent l'image des éditeurs auprès des utilisateurs mais peuvent aussi faire réagir le développeur.

Dans ce numéro, nous allons voir qu'il est (encore) possible de vivre de ses apps sur les stores, mais les conditions de cette réussite sont de plus en plus difficiles à atteindre. Nous parlerons aussi beaucoup de langages : comment créer son langage et comment celui-ci se structure, les nouveautés de PHP 5.3. Nous aborderons aussi longuement la programmation orientée modèle et particulièrement dans le domaine des télécoms.

Nous aborderons REACT, le fonctionnel et le refactoring, la programmation graphique sur Amiga, GraphQL, la programmation avancée C++ sur Linux, Powershell, ou encore un projet un peu dingue, d'un véritable cluster en Lego !

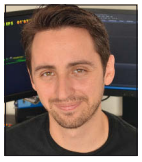
Que le code soit avec toi !

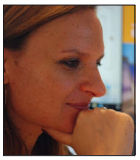
François Tonic
ftonic@programmez.com

SOMMAIRE


L'IA va-t-elle tuer le développeur ? ----4

Agenda ----6


Vivre de ses apps ? ----10



Devenir développeuse ? Un rêve ! ----16

Matériel ----19

Tableau de bord ----20


Créer son langage de programmation ----21

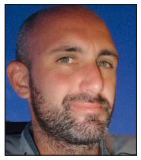
PHP 7.3 ----28


OpenStreetMap partie 1 ----34

Du temps réel avec SignalR ----39



Programmation fonctionnelle partie 1 ----41


React partie 1 ----47


GitLab partie 2 ----51


C++ & Linux ----55

Blockchain & Java partie 2 ----58


Powershell partie 1 ----62

Programmation orientée objet partie 2 ----65

GraphQL partie 2 ----69


Cluster & Lego ----73


Graphisme en Amiga partie 1 ----78

CommitStrip ----82

Abonnez-vous ! ----42

Dans le prochain numéro !

Programmez! #223, dès le 2 novembre 2018

Cybersécurité & hacking

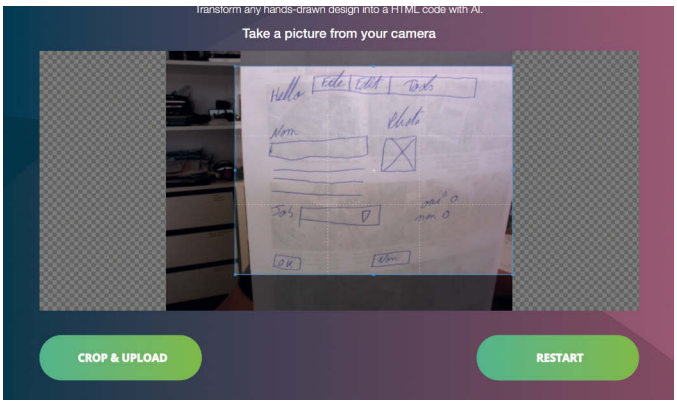
Le développeur plus que jamais acteur de la sécurité

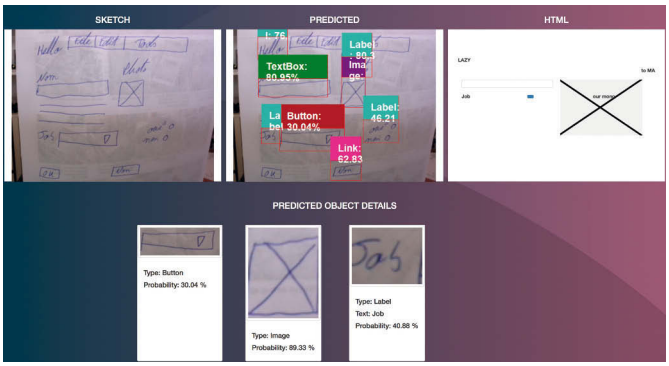
OpenStreetMap, REACT : *exemples d'implémentation*

François Tonic

Remplacer les développeurs par l'IA ?

Début septembre, c'est la grande peur ! L'IA va remplacer purement et simplement le développeur. Panique, les journaux s'emparent du sujet, souvent en exagérant considérablement les faits. Bref, le développeur sera mort dans quelques années… Tout est parti du projet de recherche Sketch2Code, que l'on peut traduire du « dessin/croquis » au code.





L'idée de Sketch2Code est très simple : à partir d'un dessin à la main, on génère du code HTML grâce à l'analyse IA. Le tout est hosté par la plateforme Azure et l'analyse se fait par les API et services IA/ML/cognitifs d'Azure.
Les étapes sont réduites à 4 :

- On charge le dessin ;
- Le service l'analyse ;
- Le service génère le résultat et montre les éléments qu'il comprend avec un % de reconnaissance ;
- On peut récupérer le code HTML généré.

Les exemples proposés sur le site montrent le potentiel. Mais on remarque aussi que les dessins sont clairs, ordonnés et structurés. Bref, plus le dessin sera bien fait, plus le % de reconnaissance sera meilleur. Nous avons essayé avec un dessin, disons fait à la va-vite sur Photoshop, la reconnaissance était assez aléatoire et les interprétations sont donc elles aussi aléatoires. Plus on se rapproche de 90 – 100 %, mieux c'est. Plus ce % est faible, moins le moteur va générer le bon objet d'interface, ou pire, comprendre ce qu'il doit en faire.
Et c'est là une des faiblesses du service : le % de reconnaissance est particulièrement variable.

L'importance d'un dessin parfait !

Quand on regarde les dessins proposés en exemple et ce que nous avons chargé, on se dit : soit les développeurs de Microsoft sont très doués en dessin, soit nous sommes très mauvais. Il faut effectivement être très rigoureux dans les formes, les textes. D'autre part, mieux vaut écrire les textes et chiffres en anglais.

Si vous rêvez d'utiliser Sketch2Code comme outil au quotidien, oubliez tout de suite. Pour le moment, le service est un démonstrateur technologique. Mais il montre ce que les technologies cognitives, les outils de Machine Learning/IA pourront apporter aux développeurs. Imaginez remplacer le développeur par ce genre de service comme nous avons pu le lire!, c'est faux et irréaliste dans l'état actuel des technologies.

Par contre, oui, cette technologie, ou une autre, pourra faciliter certaines tâches comme l'interface graphique, créer très rapidement des maquettes fonctionnelles. Il y a quelques mois, Microsoft Garage, un lab expérimental, avait sorti une app, Ink to Code, pour créer des interfaces à partir d'un dessin/schéma. Dans l'application, on dessine les objets d'interface et l'outil reconnaît les formes. Ce prototype peut être exporté vers Visual Studio : en UWP ou Xamarin Android.
Sketch2Code pousse un peu plus loin ce type de services, mais n'est pas nouveau sur le marché. Il en existe d'autres, à la qualité là aussi variable. Ainsi, SketchCode peut générer du HTML depuis un croquis. Citons encore : Anima Toolkit, Supernova.
Reste à voir la qualité de conversion et le code généré derrière. Ces outils seront viables quand le % de reconnaissance sera élevé et que la génération sera de bonne qualité avec un code propre et ne nécessitant pas de grosses modifications. Car si vous devez modifier une large partie de l'interface et du code, on perd tout l'intérêt du service et on perd du temps à refaire.

Ces nouveaux outils/services vont donc bien plus loin que les outils passés et actuels : génération de code via des modèles UML, approche low code/no code, on crée graphiquement les interfaces et le code se génère automatiquement, etc. Ces outils existent depuis de nombreuses années, avec plus ou moins de succès. Si on peut générer une partie de l'interface et du code lié, la partie fonctionnelle et la glue technique restent à créer. Mais ces outils ont le mérite de mâcher une partie du travail.
D'autres recherches étaient ouvertes sur la possibilité de génération du code à partir d'un texte. Les premiers travaux n'étaient pas forcément très concluants, car il fallait un texte parfaitement structuré avec des mots-clés et une grammaire spécifique. Ce qui limite énormément les possibilités. Cependant, l'idée n'était pas inintéressante.

Bref, la matrice ne va pas remplacer le développeur avant quelques années… •

Microsoft
experiences18

6 ET 7 NOVEMBRE 2018
PALAIS DES CONGRÉS DE PARIS

REJOIGNEZ-NOUS

ÉVÉNEMENT GRATUIT

Inscrivez-vous : aka.ms/experiences18

- Plus de 50 sessions techniques avec des démos
- Un écosytème de centaines d'experts et entreprises
- Une keynote avec des grands speakers de l'industrie

#experiences18

Microsoft

CONFÉRENCES TECHNIQUES

XebiCon par Xberia

La conférence qui vous donnera les clés pour tirer le meilleur des dernières technologies : Data, Architecture, Mobilité, DevOps, etc. 20 novembre 2018 - xebicon.fr

OCTOBRE

Les assises de la sécurité

Du 10 au 12 octobre, Monaco

La référence des évènements sécurité revient pour la 18e année ! Durant 4 jours, les conférences, ateliers, tables rondes se succèdent sur tous les thèmes autour de la cybersécurité. De multiples sessions plénières animeront les différentes journées. Les assises ne parlent pas uniquement de sécurité mais aussi de l'IT en général.
Pour en savoir plus :
https://www.lesassisesdelasecurite.com

Meilleur Dev de France

23 octobre, Espace Grande Arche, Paris – La Défense

L'évènement MDF revient pour la 6e année ! Les développeurs de toute la France vont s'affronter durant toute la journée et relever les défis de programmation. Plus de 2 000 participants, une multitude de conférences, cette journée est placée sous le signe du code, du code et du code. Venez transpirer et faire chauffer les claviers.
Programmez ! sera présent toute la journée.
Pour en savoir plus :
https://www.meilleurdevdefrance.com

Forum PHP 2018

25 & 26 octobre, Marriott Rive Gauche, Paris

L'AFUP organise la nouvelle édition de la grand'messe PHP de l'année. Le Forum PHP est l'occasion de rencontrer les meilleurs experts français et mondiaux de PHP et des outils liés, de rencontrer la communauté et de découvrir les dizaines de conférences techniques et les nombreux retours terrains. Le forum, c'est de la technique mais aussi la partie entreprise et les projets IT.
Site : https://event.afup.org

VoxxedDays 2018

Du 29 au 31 octobre, Paris

La première grande conférence sur les micro-services se tiendra fin octobre. VoxxedDays a l'ambition de dresser un état de l'art de l'architecture par micro-services. De nombreuses sessions techniques seront proposées avec des keynotes, des ateliers et des quickies. Au-delà, on parlera conteneurs, cloud et infrastructure.
Profitez de -20 % pour les lectrices et lecteurs de Programmez ! : VXDMS18_PROGRAMMEZ
Inscription :
https://tickets.voxxeddays.com/event/vxdms2018

NOVEMBRE

Microsoft Experiences18

6 & 7 novembre, Palais des congrès, Paris

C'est l'évènement annuel de toutes les communautés et des utilisateurs des technologies Microsoft. Comme chaque année, la première journée est placée sous le signe du business et comment la technologie peut aider l'entreprise, les utilisateurs. La seconde journée est placée sous le signe de la technique et des développeurs. On parlera code, outils, méthodes agiles, sécurité ! Des centaines de sessions et d'ateliers seront proposés sur les deux jours. Une occasion unique pour rencontrer d'autres développeurs et les experts Microsoft. Venez rencontrer la rédaction de Programmez !
Inscrivez-vous dès maintenant sur www.experiences18.microsoft.fr

DEVFEST TOULOUSE 2018

Date : 8 novembre, Toulouse

Le DevFest Toulouse est un événement organisé par les communautés de développeur.se.s de Toulouse, et porté administrativement par le GDG Toulouse.

Pour rendre tout cela possible, une équipe de bénévoles s'active en coulisses.
Fort des deux dernières éditions, l'équipe remet le couvert pour une troisième année. Les nouveautés 2018 : plus de talks, de speakers, dans un lieu plus grand : le centre des Congrès Pierre Baudis, 600 participants attendus … et un super thème : le rétro gaming/rétro computing !
Plus d'infos sur : https://devfesttoulouse.fr

MongoDB Europe' 18

8 novembre, Londres

La conférence européenne de MongoDB se tiendra à Londres. Une bonne occasion de rencontrer les équipes et d'aborder de nombreux thèmes : développement, cas d'usages, cloud, clustering, MongoDB par secteur d'activité.
Pour en savoir plus :
https://www.mongodb.com/europe18/agenda

Devops D-Day#4

15 novembre, Orange Vélodrome, Marseille

La journée DevOps D-Day revient pour la 4e année ! On y parle DevOps mais pas que ! Docker, les conteneurs, la transformations IT et le cloud sont les thèmes de la journée. La journée promet d'être chargée avec de nombreuses sessions et les différentes keynotes.
Site : http://2018.devops-dday.com

Maker Faire Paris

Du 23 au 25 novembre, Cité des sciences, Paris

Le grand évènement maker change de date ! Prévu initialement début novembre, la Maker Faire Paris se déroulera du 23 au 25 novembre. L'édition 2017 avait réuni plus de 800 makers et des milliers de visiteurs. Cette année encore, il s'agit de rencontrer toutes les communautés makers et DIY, assister aux conférences et les nombreuses démos !
Site : https://paris.makerfaire.com

[Programmez!] Le magazine des développeurs

prépare son premier livre sur l'Histoire de l'informatique.

Nous avons besoin de votre soutien pour générer la version 1.

Campagne de financement participatif sur ulule :

https://fr.ulule.com/histoire-informatique/

L'âge d'or de la micro-informatique

Je me souviens des premières machines que j'ai utilisées. A 10 ans, je récupère une Casio PB100. A 12 ans je découvre l'Amstrad CPC et le Macintosh. Puis l'Atari ST et un peu l'Amiga 500. Une autre époque de l'informatique.

Je vous propose une remontée dans le temps avec les plus belles machines ou que je considère comme incontournables pour comprendre l'évolution de la micro-informatique, les oppositions, les alliances et les plus gros échecs.

Deux bornes essentielles :
1973 : le premier micro-ordinateur, le Micral N
2007 : le 1er iPhone

Enjoy !

[Programmez!] Le magazine des développeurs

Une histoire de l'informatique

Les **ordinateurs** de 1973 à 2007

François Tonic
Historien, journaliste
Rédacteur en chef de
Programmez! & cloudmagazine.fr

Neler-IT

[Programmez!] Le magazine des développeurs

Campagne jusqu'au 26 octobre !

Programmez facilement vos scans et impressions sur terminaux **ZEBRA Android** avec **WINDEV**

Un nouveau composant open source pour WINDEV voit le jour pour aider les concepteurs d'applications mobiles industrielles sous matériel Zebra Android. Jamais scanner et imprimer n'a été aussi facile, jugez plutôt !

Zebra Technologies doit sa notoriété à son expertise dans la fabrication de terminaux mobiles, de tablettes, d'imprimantes, de scanners, ou encore de lecteurs RFID & NFC.
Ces solutions sont régulièrement utilisées dans les secteurs du retail, de la santé, du transport et de la logistique ou du tourisme, ce qui en fait un leader mondial.

Comment utiliser ces ressources pour Android?

Historiquement, les intégrateurs et ISV ont toujours pu bénéficier de ressources mises à disposition par Zebra.
Ainsi, la solution logicielle nommée DataWedge est intégrée en standard dans tous les périphériques Zebra, Il s'agit d'un module de type fournisseur de services, facilement configurable avant tout lancement d'un traitement de numérisation ou de lecture de codes à barres.
Alors, comment font les concepteurs d'applications mobiles pour réutiliser ces ressources ?

Nous nous sommes placés dans le contexte d'un développement avec WINDEV, le célèbre AGL édité par PC SOFT (vendu dans plus de 140 pays). Plus précisément, nous illustrerons notre cas pratique via WINDEV Mobile, clone de WINDEV dédié aux applications mobiles aussi bien pour des terminaux Windows, Android qu'iOS en natif.
Bien-entendu, WINDEV Mobile est connu pour sa richesse fonctionnelle (IDE, base de données intégrée, tests unitaires, L5G, télémétrie pour le back end…) et sa faculté à créer des applications cross-platforms natives (CE, Android, iOS).
Nous nous concentrerons ici sur l'aspect scan et impression sur des terminaux Android.

Intégration simple et rapide du scan Zebra avec WINDEV Mobile

La méthode est simple : toutes les informations utiles à l'interface avec le matériel Zebra sont disponibles en ligne sur le site Zebra.
Classiquement, Datawedge peut être contrôlé à l'aide de nombreux Intents Android facilement pilotables grâce à un insert de code Java depuis WINDEV Mobile. Le code WLangage est du type :
« DataWedgeDemarre(nom_de_la_call back_java) »

Un composant open source pour booster les développements WINDEV Mobile sous Android

Toutefois, pour rester fidèle aux gains de productivité habituels de WINDEV Mobile, un partenariat technique entre le constructeur et l'éditeur a permis de proposer encore plus de simplicité. Ainsi, un composant a été créé pour piloter encore plus facilement les terminaux mobiles. Concrètement, cela se traduit par des fonctions facilement identifiables comme *DWActiverDataWedge*, *DWDemarrerUnScan*, *DWStopperUnScan*, *DWModifierLesParametresDuScanner.*

WINDEV Mobile et Zebra : excellente impression !

Et pour les impressions, me direz-vous ? C'est pareil !
Que les imprimantes soient connectées en WiFi, Bluetooth ou USB, le développeur bénéficie de fonctions intuitives comme *ZebraPCImprimeLigne, ZebraPCImprimeTexte, ZebraPCImprimeGraphique, ZebraPCStatusImprimante.*

Concrètement, nous allons illustrer la méthode avec l'impression d'étiquettes avec codes à barres.
Lancez le logiciel *Zebra Designer* pour concevoir graphiquement l'étiquette. Cela génère la chaîne ZPL (équivalent du CPL) utile à l'impression et vous la collez dans le code WLangage
Enfin, la fonction du composant WINDEV Mobile « ZebraPCImprimeChaineZPLVariable » lance l'impression.
Et voici le résultat de l'impression sur un modèle ZQ320. (image ci-dessus)

La réalisation du template et l'écriture du code d'impression de l'étiquette ont été réalisées en moins de 5 minutes, top chrono !
Il aurait fallu des dizaines de lignes de code et beaucoup plus de temps (en heures) sans WINDEV Mobile.

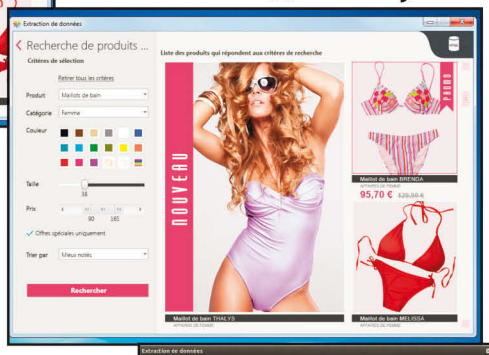
Ressources disponibles ici :
http://techdocs.zebra.com/datawedge/6-8/guide/about/
Composant WINDEV Mobile : https://depot.pcsoft.fr/

Un code et des interfaces uniques pour toutes les plateformes

WINDEV®

Application **WINDOWS**

Application **JAVA**



Application **LINUX**

WINDEV 23

DÉVELOPPEZ

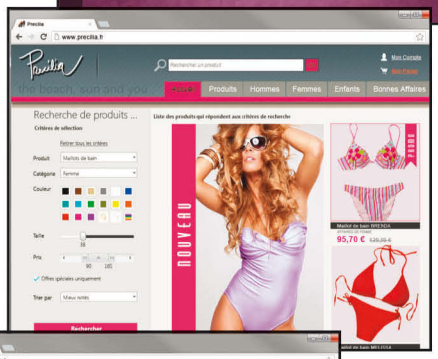
UNE SEULE FOIS,

PUIS RECOMPILEZ

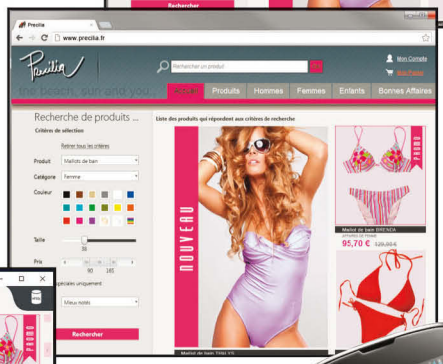
(NATIVEMENT)

POUR CHAQUE

CIBLE



Site avec serveur **LINUX** avec WEBDEV



Site en **PHP** avec WEBDEV

Application **UWP**

Application sur **Smartphone, Tablette et Terminal industriel** avec WINDEV Mobile

Avec WINDEV, WEBDEV et WINDEV Mobile 23, développez «une seule fois», et créez:

Des applications natives:
- Windows
- Linux
- Mac
- Java

+ Des sites pour :
- Windows
- Linux
- ou en PHP

+Des applications mobiles natives pour smartphones, tablettes et terminaux industriels:
- Android
- iOS
- UWP
- Windows CE.

Tout est natif.

WINDEV élu
«Langage le plus productif du marché»

VERSION EXPRESS GRATUITE
Téléchargez-la !

WINDEV VU À LA TV
SUR **TF1, M6, BFM TV, CNEWS, LCI**

Tél : 04 67 032 032

WWW.PCSOFT.FR

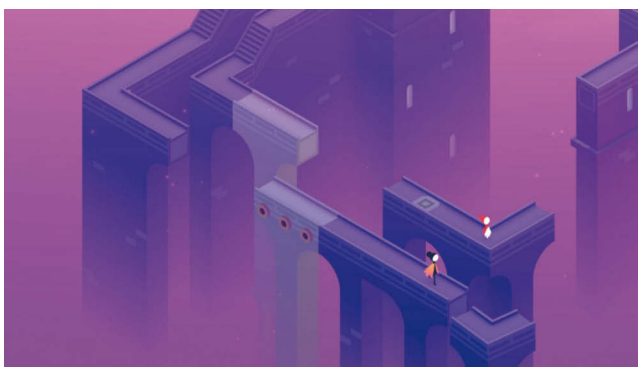
François Tonic

Les développeurs vivent-ils de leurs apps sur les App Stores ?

En 2008, Apple présentait ce qui allait être la révolution du smartphone : l'App Store. Ce modèle entièrement mobile allait faire exploser les modèles de distribution et de consommation des applications. L'App Store a forcé tous les autres acteurs à bouger. Aujourd'hui, App Store et Play sont les deux mastodontes des boutiques d'apps. D'ailleurs, Apple se vante d'avoir distribué aux développeurs 100 milliards $. Le développeur gagne-t-il réellement de l'argent sur un store ? Comme toujours, il y a une énorme disparité entre le développeur bankable et les autres. Explications.

Cette question est parfaitement légitime, car aujourd'hui, les apps mobiles envahissent tous les domaines. Et ce sont plusieurs millions de développeurs qui développent et publient sur les différents stores. Selon des chiffres d'Apple, printemps 2018, il y aurait 332 000 développeurs français inscrits. Ce qui induit une concurrence féroce, particulièrement sur les jeux, qui reste la rubrique la plus active.

Deux éditeurs résument la situation (pour les studios installés)



Tout le monde ne publie pas un Monument Valley chaque jour. Le (petit) studio ustwo a eu la géniale idée de ce jeu, mais il s'en est donné les moyens : 1 an de développement, 850 000 $ de budget. Résultat : après 1 an, le jeu avait rapporté presque 6 millions $. Monument Valley 2 a rapporté +10 millions $ pour un budget de 2 millions $! Mais l'éditeur a eu le réflexe de ne pas exploiter outrageusement son jeu. Au contraire, l'équipe a limité les extensions et les versions.

Cette approche est à l'opposé de Rovio. Rovio est l'éditeur du cultissime Angry Bird. Le succès a été immédiat et dans le monde entier. Rovio a décliné son jeu en x éditions, avec de plus en plus d'achats intégrés, des gameplays douteux et sans réel renouvellement du concept. Surtout l'éditeur n'a pas su trouver d'autres jeux pour palier la baisse régulière d'Angry Bird. Résultat : licenciement, fermeture de bureaux, baisse en bourse, etc.

Dans ce marché, tout va très vite. Bien entendu, tout le monde n'a pas un studio ni le succès planétaire des deux studios cités. Les petits éditeurs ou les indépendants seront aussi plus sensibles aux politiques des éditeurs des stores, aux changements de règles, etc. Et la plupart du temps, le développeur devra suivre ou retirer son app. Et il est de plus en plus difficile de se faire une place. Les gros studios occupent la place et les mécanismes des stores sont parfois obscurs, par exemple, sur les mises en avant.

> « Prix de vente
> – commission du store
> – TVA
> = ce que je touche réellement »

Parfois le succès vient de la visite du patron d'Apple dans les locaux de VizEat qui va donner une visibilité énorme, ou encore mieux, être invité à montrer son app sur scène durant une keynote. Si tout se passe bien, vous toucherez plusieurs millions de personnes en quelques minutes, sans compter le replay.

Mais revenons à notre sujet initial : peux-tu vivre de ton app sur les stores ?

Une révolte en cours ?

Fin août, plusieurs sites et éditeurs ne veulent plus des commissions prises par Apple et Google. Quand Netflix, Spotify et l'éditeur de Fortnite commencent à s'agiter, immédiatement, on sent que quelque chose se passe. En réalité, la situation se dégrade depuis plusieurs mois. Les commissions ne sont qu'un des reproches : conditions générales peu claires, validation et refus parfois obscurs, mise en avant au bon vouloir du store. Pour Netflix, il s'agit de tester, sur une période limitée et des pays ciblés et pour les nouveaux clients, un nouveau mode d'abonnement en se passant de l'App Store. L'objectif est ne pas payer les commissions. Le service avait fait de même pour Play au printemps dernier.

Des gouvernements regardent aussi de près les conditions générales des stores, parfois en les accusant de monopoles ou d'imposer les conditions aux développeurs. Et ceux-ci réagissent aussi de plus en plus fortement contre ce diktat (réel ou supposé) des éditeurs.

C'est tout le paradoxe de cette affaire : les stores ne peuvent pas vivre sans les développeurs et les développeurs (du moins une partie) ne peuvent vivre sans les stores. Un équilibre doit être trouvé.

Je veux me lancer !

Si votre motivation est de vivre de vos apps, le point de départ sera (oh surprise) : l'idée. Mais une idée, aussi bonne soit-elle, n'est qu'un des éléments de votre succès, ou de votre échec.

Avant même de parler technique, il faut définir votre projet :

- Mon app peut-elle répondre à une demande, un besoin, un nouveau marché ?
- Des utilisateurs sont-ils prêts à payer x € et si oui, quel prix définir ?

Bien connaître les cibles matérielles / systèmes

Que ce soit sur Android ou iOS, vous devrez déterminer le niveau de compatibilité de votre app :

- Les modèles de smartphones / tablettes que vous voulez supporter ;
- La version d'OS minimale.

Ces deux éléments vont déterminer le niveau de fonctionnement et une partie de vos outils et SDK. D'autre part, analysez bien les données des constructeurs pour déterminer les OS les plus répandus, la taille d'écran utilisé, le pays des utilisateurs, etc. Si vos utilisateurs sont aussi à l'étranger, il faudra prévoir la localisation.

Déterminer aussi les fonctions utilisées dans votre app. Vérifiez qu'elles soient disponibles sur les configurations cibles. C'est un peu bête de le répéter, mais mieux vaut prévenir que de tout recoder !

Sur les stores Windows et macOS, généralement, cela se limitera à la version d'OS et à la configuration minimum.

Bref : attention à la fragmentation des plateformes. Ne partez pas sur les dernières technologies si le parc installé reste faible.

- Quel modèle économique choisir ? (achat, achats intégrés, abonnement).
- S'il s'agit d'un jeu, est-ce que mon idée existe déjà ? Est-ce que je suis une tendance ?
- Quelles zones géographiques ? Quels langages vais-je supporter ?
- Est-ce que je peux définir un budget prévisionnel ?
- Ai-je besoin d'un statut légal pour vendre mes apps et donc recevoir l'argent généré ?

Si on parle un peu technique, là encore, de nombreuses questions se posent :

- Quel(s) app store(s) je veux supporter ?
- Selon le choix, est-ce que je suis formé / compétent à développer dessus ?
- Si non, comment me former, comment me perfectionner ?
- Est-ce que je suis équipé (matériel + outillage) ?

JÉRÉMY MOUZIN : retour du développeur d'Hercule

Jérémy est développeur d'apps sur Android. Il a quitté l'entreprise où il travaillait pour se lancer dans les apps et en vivre. Son app la plus connue est Hercules. Il évoque son parcours et la difficulté à se faire une place dans les stores.

Tu proposes sur Google Play ton app Hercules. Comment l'idée t'es venue ? Est-ce que tu avais une idée précise de ce que tu voulais faire ?

Je voulais trouver une app où je puisse rentrer mon programme et que l'appli "déroule" celui-ci avec le minimum d'interactions possibles (juste dire à l'app que j'ai fini l'exercice). À l'époque je faisais la méthode lafay et avoir une séance complète préfédinie me plaisait bien. J'ai cherché pendant des jours une app comme ça et je ne trouvais rien, que des apps où il faut à chaque fois indiquer l'exercice qu'on fait et ses reps et charges avant de le faire. Insupportable ! J'étais en phase de recherche d'une idée d'app, car je voulais créer mon entreprise et donc j'ai simplement créé ce que j'aurais aimé trouver sur le store.

Au départ, tu n'étais pas développeur Android, il a fallu te former. iOS n'a jamais été une option ?

Je connaissais très bien le JAVA, car j'avais déjà travaillé avec ce langage par le passé, or pour iOS, à l'époque c'était l'Objectif-C (maintenant c'est Swift) qu'il aurait fallu que j'apprenne en plus d'apprendre à créer une app mobile. Pour me permettre d'aller plus vite dans le développement, j'ai préféré choisir Android et JAVA. Aujourd'hui je sais que mes revenus auraient pu être bien meilleurs en étant aussi sur iOS, mais en même temps la concurrence est plus rude, donc je ne saurai jamais si j'ai fait le bon choix.

Avais-tu un plan pour vivre de tes apps comme Hercules ? Ou te lancer dans l'aventure et que tu verrais ce que cela donnerait ?

Je savais que je voulais gagner de l'argent avec mon app pour en vivre et créer un revenu passif, mais je n'avais aucune idée de comment j'allais pouvoir générer des revenus. A priori je partais sur un modèle classique d'app gratuite avec de la pub, et je me suis rendu compte que ça ne rapportait pas du tout et puis je déteste la pub. J'ai donc passé l'appli en payant après une période d'essai gratuite de 7 jours.

Qu'est-ce qui est le plus difficile / démotivant quand un développeur veut vivre de ses apps sur un app store ?

Il y a beaucoup de choses, déjà, la perception qu'ont les gens de la valeur d'une app mobile. Payer 10 € une app mobile c'est la ruine, alors que ça coûte très très cher de la développer. Ensuite, il y a les commissions Google de 30% et la TVA qu'on perd également. Ce qui fait que sur le prix de vente, il ne reste que 58% du prix d'achat payé par les utilisateurs. Il y a beaucoup de concurrence partout et sur toutes les plateformes, donc pour percer, il faut investir énormément de temps et d'argent dans le projet, c'est très long. Enfin la pire chose c'est le manque de visibilité des nouvelles apps sur le store. Il faut nécessairement faire du marketing en ligne et donc investir du temps ou de l'argent pour faire connaître l'app, sinon ça ne sert à rien et personne n'est au courant que votre app existe !

Il y a un an, tu donnais les chiffres de tes ventes sur Play, entre 2160 et 2900 € par mois (commission déduite). Comment ce chiffre a évolué depuis ? Comment analyses-tu ces résultats aujourd'hui ?

Beaucoup de nouveaux entrants sont arrivés depuis et font de la pub à gogo sur le store, donc mes revenus ont chuté et se sont stabilisés autour de 1600 € / mois commission et TVA déduite. Avant quand on tapait "musculation" sur le play store français j'étais premier ou deuxième et il n'y avait pas de pub. Aujourd'hui je suis 4e, car 2 apps sont classées en tête, car ils font de la pub. Il suffit d'avoir un gros budget pour "récupérer" tous les utilisateurs d'une recherche sur le play store. Avec mes petits revenus je ne fais pas le poids ! Donc je me contente de ce qu'il reste. Si vous n'êtes pas dans les 5 premiers de la recherche il ne faut pas espérer avoir de trafic...

De nombreux développeurs mobiles disent qu'il est de plus en plus difficile de se faire une place sur les stores notamment à cause des clauses qui changent, le processus de validation obscur et des mises en avant de plus en plus rares. Tu en penses quoi ?

Il n'y a pas de processus de validation sur le Play Store, on met à jour quand on veut, contrairement à l'App Store d'Apple. J'ai vu que les clauses se durcissent de plus en plus et le côté technique devient de plus en plus difficile également, ce qui fait que de nos jours si on veut se lancer dans ce domaine, il vaut mieux être plusieurs et avoir du budget. Le temps de la "ruée vers l'or" est terminé, les gros budgets dictent leur loi et les petits indépendants se font écraser au fur et à mesure. Il n'y a que dans le domaine du jeu vidéo où il existe une section "indie" sur le Play Store ce qui est une bonne chose pour faire ressortir les jeux créatifs et originaux. Pour le reste c'est plus compliqué de percer à l'heure actuelle, ou du moins ça prend beaucoup plus de temps. En tout cas ce sera toujours de plus en plus dur de se faire une place sur le marché qui est maintenant saturé de partout !

Lien de l'app : https://play.google.com/store/apps/details?id=com.ingeniooz.hercule

- Est-ce que je suis référencé sur le ou les stores ? (si besoin, il faudra payer l'adhésion au programme développeur et pour pouvoir publier des apps).

Il vous faudra avoir une estimation du coût de votre développement, si vous ne faites pas une app uniquement pour le fun. Plusieurs éléments sont à considérer :

- Le matériel : l'amortissement, généralement sur 3 à 5 ans. Si besoin, y inclure le coût des outils.
- Les coûts du programme développeur pour déployer les apps.
- Électricité.
- Le temps passé à développer.

Une grosse partie du budget de votre app sera le coût du développement. Par exemple, si on prend un tarif journalier moyen de 400 € et que votre développement nécessite 30 jours (1 jour = 8h de travail), le coût du dév s'élèvera à 12 000 €, sans y inclure les différents coûts évoqués plus haut.

Même si le budget de développement est approximatif, il donne tout de même une précieuse indication sur le chiffre d'affaires que votre app doit générer pour rentabiliser votre temps. Pour un petit studio, cela inclura le développeur, le graphiste, le musicien, l'administratif, etc.

Autre point à ne pas oublier : la structure légale. Eh oui, c'est un élément qu'il faut réaliser.

Et bien entendu, regardez la rentabilité en brut et en net : ne jamais oublier les commissions des stores.

Et donc peut-on vivre de son app ?

Oui en 2018 il est toujours possible de vivre de ses apps, à la condition de bien cibler ou de profiter d'une tendance. Mais il faut être lucide : pour être dans le top des apps de sa catégorie, il faut soit un excellent retour des utilisateurs, soit investir du temps et de l'argent. Il faut miser sur le bouche à oreille de vos utilisateurs, sur les réseaux sociaux, etc. Développer une app n'est ni simple, ni rapide. Si vous voulez réussir, à la publier, ne parlons même pas d'en vivre à ce stade, il vous faudra de la patience et de la persévérance. Si vous connaissez des développeurs qui ont des apps, n'hésitez pas à discuter avec eux et avoir un retour d'expérience.

A vous de jouer !

En effet, l'app est disponible gratuitement sur le store. Pour pouvoir utiliser l'app, les utilisateurs doivent souscrire à un plan annuel. L'application est en quelque sorte louée par les utilisateurs, c'est un modèle de services que l'on voit un peu partout. Cette approche permet d'avoir des revenus réguliers. Avant, les utilisateurs devaient souscrire un support annuel, mais cette approche ne permet pas de lisser les revenus.

Stéphane ne reverse aucune commission au Store. Et il n'y a aucun achat intégré. Est-ce légal ? « *Oui c'est totalement légal. Je respecte les guidelines* ». Précise-t-il.

L'aspect économique est important, car l'app utilise les services Azure. « *Il y a un coût récurrent à prendre en compte. La location applicative permet de supporter ce coût. Par an, les ressources Azure représentent environ 20 000 €. Cela peut paraître un gros budget, mais il faut tout de même relativiser par rapport à l'activité globale* ».

Pourquoi le Windows Store ?

Comme dit plus haut, Windows était déjà la plateforme de Stéphane. Et la cible des utilisateurs est majoritairement Windows. Stéphane vise les magasins et les entreprises, il connaît donc bien les environnements cibles. Le coeur de l'app est codé en Xamarin. « Il me sera relativement facile de porter l'app sur Android et iOS. Le principal travail se ferait sur les interfaces. Ce projet est prévu, car nous avons des demandes. » poursuit Stéphane.

Si son business model ne pose pas de souci sur le store Windows, sur iOS, par exemple, il faudra sans doute regarder de près les clauses, car Apple est plus strict.

Son expérience du store est plutôt bonne. « *Auparavant, il fallait plus d'une semaine pour déployer une mise à jour ou une nouvelle app. Aujourd'hui, c'est plus rapide. J'ai un rejet, à cause d'une routine graphique.* » Précise le développeur.

Stéphane avoue aussi que le choix du Windows Store permet d'être dans sa zone de confort. Android ? Il faut s'y mettre, mais le matériel est moins cher que sur Apple. « Un des problèmes sur Android est la fragmentation de la plateforme : trop de versions systèmes différentes, trop de tailles d'écran, etc. Mais je constate la même chose sur Windows 10. Quelle version prendre ? On essaie de laisser un an aux utilisateurs pour mettre à jour. » commente Stéphane.

Est-ce rentable ?

« Oui ! » Mais le succès dépend aussi du marketing et de la promotion de son app. « *Le succès dépend du client, que le produit soit cher ou pas. Nous ne faisons pas de promotion sur le store, mais sur les réseaux sociaux* » recadre Stéphane.

Aujourd'hui, l'app est proposée en direct sur le store, mais aussi en indirect par des partenaires. Ces derniers représentaient environ 50 % de l'activité. Et ils reçoivent une commission sur les ventes.

Pas tout à fait seul

Stéphane est le core developer de l'app. Mais en cas de besoins, plusieurs développeurs indépendants peuvent l'aider. Sur la partie support, trois personnes s'en occupent dans l'entreprise, mais le support est limité, car Stéphane a misé sur une riche documentation et des tutoriels pour aider l'utilisateur.

Mais l'app doit aussi évoluer et proposer de nouvelles fonctionnalités. Actuellement, l'app est en français, mais deux nouvelles langues seront prochainement disponibles. « *Le plus long n'est pas de traduire l'interface, mais la documentation et surtout avoir un support multilingue !* » précise d'emblée Stéphane. Parmi les développements en cours, une fonction d'écran déporté avec intégration de la publicité dynamique.

STÉPHANE VIDOUSE, développeur depuis 22 ans

Stéphane développe depuis plus de 20 ans. Son app de gestion de caisse, Cassio, existe historiquement sur environnement Windows et basé sur Win32. C'est tout naturellement qu'il la propose sur le Windows Store. Mais il a opté pour un modèle économique différent. « Mon modèle business ne dépend pas du store. C'est juste une plateforme de déploiement. » Commente-t-il.

DevCon #7

13/Décembre/2018

INFRASTRUCTURE AS CODE

Pourquoi coder son infrastructure ?

- 2 KEYNOTES
- 4 SESSIONS TECHNIQUES
- 2 QUICKIES
- 1 PIZZA PARTY

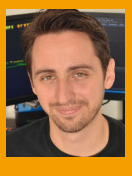
Conférence organisée par

[Programmez!] Le magazine des développeurs

Où ?

42

INFORMATIONS & INSCRIPTION SUR WWW.PROGRAMMEZ.COM



Stéphane Graziano
Webrox SARL

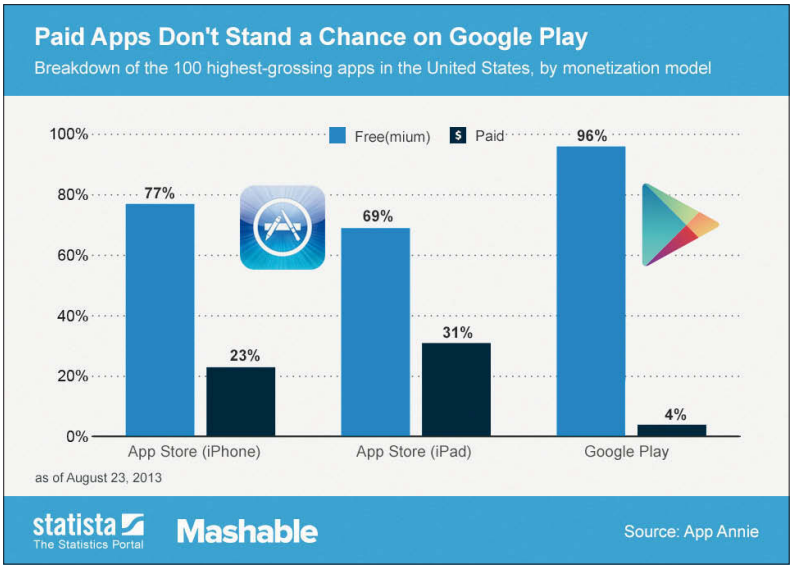
Les développeurs d'apps peuvent-ils vivre de leurs apps ?

Quand un développeur indépendant d'applications rencontre des inconnus et qu'il annonce fièrement qu'il développe des apps (pour 2,5 milliards d'utilisateurs de smartphones), la réaction des inconnus est souvent la même, et vous n'y échapperez pas : "Mais vous arrivez à en vivre ???".

Le ton est donné: ils n'achètent pas d'applications (c'est gratuit non ?), pas de quoi faire un salaire d'après ces inconnus qui sont pourtant bien équipés de smartphones dernière génération. Sans compter que ledit smartphone est bien souvent une extension de leur propre vie depuis plusieurs années.

Alors le développeur d'application, souvent peu considéré dans l'hexagone, peut-il vivre de ses créations sur les Stores ?

Note : le Windows Store de Microsoft n'a volontairement pas été intégré dans cet article puisque son heure a sonné, nous sommes nombreux à avoir subi leur mise à mort alors si vous envisagiez cette plateforme : fuyez !



CLIENTÈLE

Le monde du shareware (logiciels en version d'essais) et freeware (logiciels gratuits) inclus dans les CD des magazines a également bercé tous les informaticiens dans les 90's, et a permis à beaucoup d'indépendants de se faire connaître, mais ça n'avait pas encore atteint le grand public si profondément. Apple a mis sur le marché un nouveau modèle économique en 2007 d'apps : des petites fonctionnalités à des prix mini (gratuité ou quelques euros).

Rappelez-vous de votre premier smartphone, vous passiez votre temps sur les Stores en quête de nouvelles applis gratuites super pratiques ou des jeux tout aussi gratuits pour passer le temps.

De 2007 à aujourd'hui, une pluie d'application s'est déversée dans les Stores, on en compte 2 millions sur l'AppStore Apple et 3.5 millions sur GooglePlay. 100 millions de dollars sont dépensés chaque jour sur l'AppStore ! Ces chiffres gargantuesques bien relayés par les médias donnent le tournis et pourraient convaincre n'importe quel développeur de se lancer dans l'aventure.

COMMISSIONS DES STORES

Les Stores prennent une commission de 30% sur le prix HT, et s'occupent par contre de reverser la TVA de 20% aux administrations. Vous toucherez donc 58% net dans votre poche du prix fixé.

Pour une appli à 1€ on vous payera 0,58€. Ça aurait pu s'arrêter là mais vous n'aurez pas encore payé vos charges sur salaire, suivant votre structure on peut considérer que vous pourrez vous payer environ 0,30€.

PUBLICITÉ

La régie de publicité de Google "AdSense" a été adaptée pour le marché mobile "AdMob" et il n'y a pas vraiment d'alternative aussi rentable et pérenne. Les revenus sont extrêmement variables suivant les campagnes; on peut espérer en moyenne de 0.50€ à 1€ pour 1000 impressions. Le volume d'affichage pour dégager un salaire sera donc entre 1,5 million et 3 millions de publicités par mois. En général, il faut voir la publicité comme un complément et pas comme une source de revenus principale, mais si vos utilisateurs sont majoritairement jeunes, misez tout dessus.

TEMPS DE DÉVELOPPEMENT

Il n'y a bien souvent pas d'investissement significatif pour développer une application, seul le temps sera votre jauge. Une application peut prendre une année à être réalisé, une autre 1 mois et celle qui a pris un mois peut rapporter 100 fois plus que le premier.

Il n'y a pas vraiment de règles dans le monde de la création, mais mon conseil serait de sortir vos apps le plus rapidement possible pour dégager du cash et faire des mises à jour au fur et à mesure. Certaines fonctionnalités ne sont pas vitales et elles pourraient accaparer une grande partie de

votre temps de développement. Il y a beaucoup d'utilisateurs qui aiment participer au cycle de vie de l'application par mail ou twitter, ils seront ravis de voir leurs idées implémentées ou leurs bugs signalés qui auront été corrigés.
Ne sous-estimez pas le temps passé à comprendre les comportements des Frameworks, surtout quand ils évoluent de manière incessante et sont très mal documentés.

SOUMISSIONS

Ne soyez pas surpris, votre première soumission sera probablement un échec. Préparez bien 1 à 2 semaines en plus dans votre planning pour régler ça. Les Stores ont mis des process très rigoureux de vérification en place, avec, derrière, des humains à la sensibilité différente et avec qui vous ne pourrez pas communiquer en direct. Voilà un cycle de soumission possible :

- Envoi d'une nouvelle application,
- Attendre 2 à 3 jours,
- Echec de la validation,
- Envoi correctif,
- Attendre 2 à 3 jours,
- Echec de la validation car autre problème,
- Envoi correctif,
- Attendre 2 à 3 jours,
- Validation.

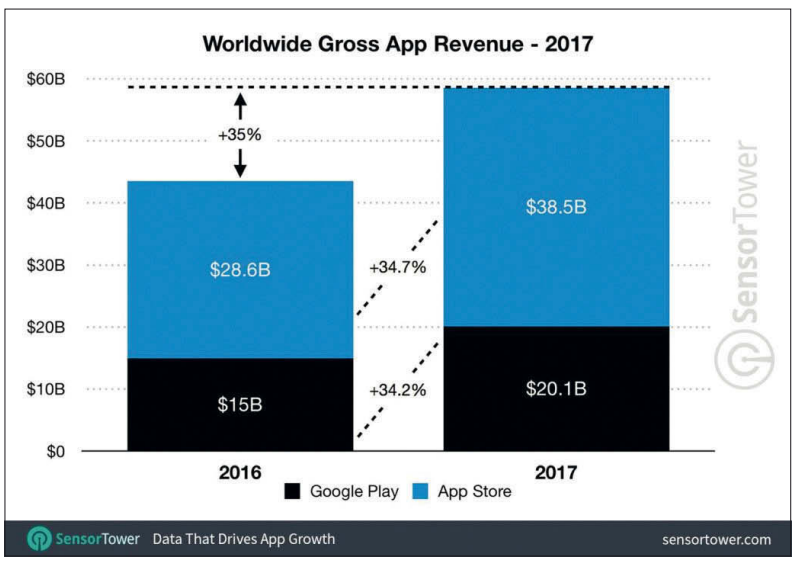
Ce petit jeu peut durer un petit moment donc soyez préparés psychologiquement à pester derrière votre écran. Si je peux vous donner un conseil, fermez les yeux, respirez un bon coup et faites rigoureusement ce qu'on vous demande car il n'y a pas de négociation possible.

SUPPORT

On l'oublie souvent mais une application qui fonctionne apporte aussi son lot de support mail. Vous devrez trouver du temps pour répondre aux nombreuses demandes. Ca peut aller du faux au vrai bug, en passant par des insultes ou plus rarement des remerciements. La recherche de bug est une étape chronophage, les mails sont souvent composés de ces 4 mots magiques "ça ne marche pas", et il faudra donc établir un dialogue avec l'interlocuteur pour qu'il détaille ses maux. Certains bugs étant liés au matériel lui-même, ça sera d'autant plus difficile d'y apporter un correctif si vous ne le possédez pas. Si vous le pouvez, faites en sorte qu'il puisse envoyer vos logs, et encore mieux que les logs soient directement adressés sur votre serveur.

CONCURRENCE

La concurrence peut-être très rude si votre application commence à être connue, qu'on peut la reproduire et que votre fonctionnalité de base est payante (achat intégré). Un étudiant pourra s'amuser à faire la même chose que votre appli payante, il la mettra gratuite sur le Store, juste pour le fun et pour la beauté du geste. Un employé pourra ouvrir une auto-entreprise en quelques clics et venir casser le marché pour se faire des compléments de salaire. Pire, un concurrent, dans la même situation que vous, pourra aussi vous voler votre idée et baisser les prix jusqu'à ce que l'un des deux suffoque (Au bonheur des dames - Zola).
La facilité d'accès aux Stores en fait aussi sa faiblesse, il faut souvent innover pour garder une longueur d'avance sur ses concurrents. Méfiez-vous également du reverse engineering, vos sources sur Android sont plus que visibles moyennant peu d'efforts, il n'est pas impossible de cloner une application. Certaines fonctionnalités des environnements de développement permettent de sécuriser vos sources : apprenez à les utiliser.



BONNE PRATIQUE

- Sortir un produit original ou avec une fonctionnalité que les autres apps ne font pas (ou pas bien) ;
- Sortir vos apps le plus rapidement possible et faire des mises à jour au fur et à mesure pour pouvoir dégager du cash et voir si votre app plaît ;
- Rester calme quand vos apps échouent la validation des Stores et effectuer à la lettre ce qu'on vous demande.

MAUVAISE PRATIQUE

- Inonder vos apps de publicité ;
- Rendre votre application payante sans version d'essai ;
- Mettre des prix excessifs dans un marché concurrentiel.

RENTABILITÉ

Après avoir fait le développement, géré les bugs et le support, passé l'étape des Stores, avoir été amputé de 30% de commission, on peut se demander si c'est rentable vu la somme colossale de choses à connaître pour de si petites applications.
A moins d'avoir créé une appli à succès, le constat est plus que mitigé, malgré ce que peuvent faire miroiter les médias ; on peut compléter ses revenus, une poignée pourront en vivre et seuls peu d'élus décrochent le jackpot. D'ailleurs la success story passera bien souvent par la case investisseurs pour avoir des fonds (en publicité pour se faire connaître par exemple). Les grands noms du marché actuel ont en effet tous été catapultés grâce à de gros investissements, la route est longue!
Tout le monde a déjà eu une idée révolutionnaire d'application, la mise en oeuvre et la commercialisation sont deux mondes bien distincts, il ne faudra jamais l'oublier.

Tubecast (bitly.com/tubecast)
LAN drive (bitly.com/landriveinstall)
Poppy Kart 1 (bit.ly/poppykartinstall)
Poppy Kart 2 (bit.ly/poppykart2)

Yaël DEHAESE
Créatrice de It4Girls
@IT_4_Girls

Thierry LERICHE
Architecte pour SG
@ThierryLeriche

Quand je serai grande, je serai développeuse

Le constat est clair. Les femmes représentent moins d'un cinquième des effectifs dans les métiers techniques de l'IT. À l'occasion d'une semaine dédiée à la Tech dans une grande banque, nous avons croisé Yaël Dehaese créatrice d'It4Girls, en pleine séance d'initiation à la programmation avec un groupe d'enfants. Elle a volontiers accepté de nous présenter cette initiative.

Que propose la start'up It4Girls ? Comment se démarque-t-elle ?

Nous organisons des ateliers de découverte de la programmation à l'aide d'outils adaptés aux enfants. Cela ressemble beaucoup à DEVOXX for Kids ou Programatoo, entre autres, mais en ciblant plus spécifiquement les filles.

Nous avions nous aussi organisé, au sein de l'entreprise qui nous incube, plusieurs ateliers de coding. Or nous avions constaté que les garçons s'y inscrivaient plus volontiers que les filles alors que les ateliers étaient ouverts à tous. Ils représentaient deux tiers des effectifs, quels que soient les organismes ou sociétés qui proposaient ces ateliers. Nous voulions inverser la tendance et attirer aussi les filles. Du coup, nous avons conçu des ateliers en adéquation avec leurs attentes.

Par exemple, nous leur proposons de créer des jeux qui se rapprochent le plus possible de leurs couleurs préférées, de leurs animaux fétiches ou encore de leurs héroïnes favorites. Pour autant, nous ne tombons pas forcément dans un jeu où une licorne à paillette doit trouver la sortie d'un labyrinthe rose peuplé de fées et de dauphins…

Atelier avec un groupe de jeunes filles

> *« malheureusement ça ne va pas en s'améliorant… »*

Pourquoi n'inviter que des filles ?

On constate que les petites filles ne s'intéressent pas plus que ça aux ordinateurs et, par extensions, rares sont les parents qui leur proposent une activité liée à l'informatique. Mais, lorsqu'on explique aux parents que c'est un atelier destiné aux filles, avec un thème adapté, ils inscrivent plus volontiers leurs enfants que lorsque c'est mixte.

Le thème séduit les filles. Nous leur proposons de créer un jeu avec des chevaux et des poneys qu'elles pourront personnaliser, des films d'animation avec les héroïnes de leur choix ou encore un animal virtuel qu'elles pourront nourrir et divertir.

Nous accueillons parfois aussi des garçons (les frères par exemple) ou lorsqu'un parent souhaite malgré tout inscrire son fils, même si on verra peu de ninjas ou de monstres dans l'atelier.

Quelle est la proportion de femmes dans l'IT ?

De nos jours, on constate que le nombre de femmes travaillant dans les métiers de l'informatique et de la technologie oscille entre 25 et 30%. Ce chiffre inclut néanmoins des métiers comme Chef de Projet ou Managers d'équipe. Lorsqu'on considère des équipes purement techniques, composées de développeurs ou de techniciens réseau par exemple, la proportion de femmes tombe à 5%.

Les femmes ont toujours travaillé dans l'informatique mais, lorsque ce secteur s'est développé dans les années 80 puis 90, et que de nombreuses écoles se sont créées, les garçons ont massivement rejoint cette filière. Le nombre de filles a alors peu évolué tandis que celui des garçons est monté en flèche. En proportion, les filles ont donc quasi disparu.

Le manque de filles a créé un cercle vicieux : personne n'ayant envie de se retrouver en minorité dans un groupe, à un âge où le besoin d'intégration est fort, c'est devenu plus compliqué pour une fille de rejoindre cette filière.

Ce phénomène s'observe dans la plupart des pays développés. Dans les pays occidentaux, les filles ont le sentiment que ce monde (informatique, jeux vidéo, nouvelles technologies) n'est pas le leur mais celui des garçons, qui ne les considèrent d'ailleurs pas toujours d'un très bon œil. En Asie et au Moyen-Orient, l'informatique n'est pas perçue de la même façon. Il n'y a

Interview d'un des enfants

Nous avons croisé un groupe de jeunes filles à la fin d'une séance. Elles arboraient toutes un large sourire. Manon, 8 ans, nous raconte cet atelier.

Qu'as-tu fait durant l'atelier ?
J'ai créé un jeu vidéo de labyrinthe sur l'ordinateur. Il y a une coccinelle qui veut manger une pomme. On déplace la coccinelle avec les flèches. Et quand on touche la pomme, on gagne un point et ensuite l'ordinateur déplace la pomme au hasard. On peut mettre d'autres fruits aussi, mais moi je préfère les pommes. À la fin, il y avait un goûter. Et quand mon papa est revenu me chercher, je lui ai montré et il a dit que c'était bien.

Et c'était comment ? Voudras-tu recommencer une autre fois ?
Oui c'était super bien. Je me suis bien amusée. La dame était très gentille. Elle nous a montré comment il faut faire. Moi j'ai choisi la musique. La fille assise à côté de moi, c'est la seconde fois qu'elle vient. Et moi je voudrais bien revenir une autre fois parce que c'était trop court (la séance a duré deux heures).

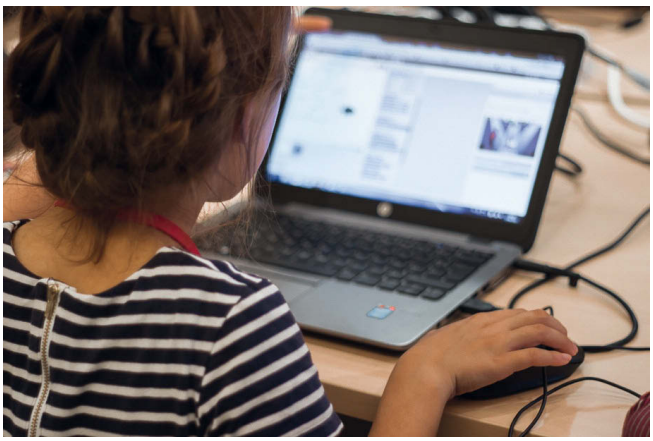
Est-ce que ça t'a donné envie de faire de l'informatique ?
Je voudrais bien continuer à la maison. J'ai déjà une idée pour un autre jeu. La dame a montré à mon papa comment il faut faire (tout se fait en ligne) et il a dit que c'est d'accord.

Programmation d'un jeu SCRATCH

pas les mêmes mythes et croyances autour de l'ordinateur, si bien qu'il y a autant d'hommes que de femmes dans l'IT. Dans plusieurs pays asiatiques, les femmes sont même légèrement majoritaires.

Comment se déroule une séance avec les enfants ?
Ce sont des ateliers « d'initiation » donc, en général, nous avons des enfants qui ne connaissent pas le langage SCRATCH et nous commençons alors par une petite démo. Ensuite nous parlons du jeu que nous allons construire en questionnant les enfants. Enfin, nous créons le jeu ensemble.
Au bout d'une heure et demie, les enfants ont réalisé le programme correspondant au jeu proposé et y jouent. Certains améliorent le jeu ou en modifient les options. Et puis vient le moment du goûter.
Quand les parents reviennent, la plupart des fillettes ne veulent pas quitter leur PC et demandent à rester dans la salle pour continuer à jouer ou à coder. Nous savons aussi que plus de la moitié d'entre elles retourneront sur Scratch à la maison : mission accomplie pour nous.

À partir de quel âge peut-on participer ?
Nous accueillons les enfants dès qu'ils savent lire. Pour un atelier parents-enfants, on commence à partir de six ans. Pour les ateliers en solo, c'est à partir de huit ans.
Nous proposons aussi aux plus grands (14-16 ans) d'apprendre les bases de la programmation à travers un jeu plus complexe et nous leur demandons ensuite d'en inventer un avec un contenu qui pourrait plaire à tous. Plutôt qu'un énième jeu de courses de voitures, de ninjas ou de destruction, nous proposons aux participants de travailler autour de la collaboration.
Là, nous sollicitons des garçons et des filles car l'idée est de créer des jeux pour la startup IT4Girls et nous sommes ravies d'associer des garçons à notre démarche (et eux aussi). Enfin, nous faisons aussi



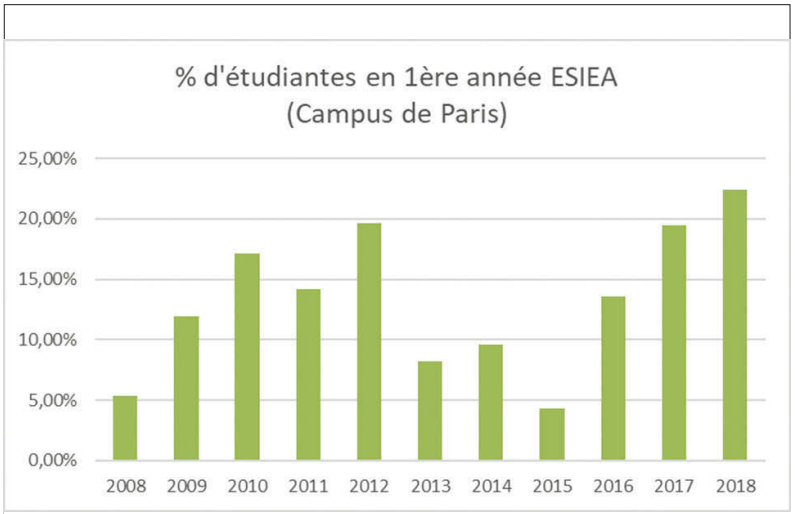
Une jeune fille en pleine création

coder les mamans afin de construire des jeux éducatifs pour leurs enfants.

« l'atelier se remplit en moins d'une demi-heure… »

Quel est le modèle économique d'IT4Girls ? Le succès est-il au rendez-vous ?

Nous avons énormément de demandes. Lorsque nous proposons un atelier à un département (environ 150 personnes), nous n'avons malheureusement que dix places et l'atelier se remplit en moins d'une demi-heure… Nous inscrivons les demandes supplémentaires sur listes d'attentes pour les ateliers suivants. À l'avenir, le challenge serait de pouvoir convier les enfants une seconde fois, voire de proposer un « parcours » sur Scratch avec des niveaux et un mini-diplôme. C'est toujours gratuit pour les enfants et leurs parents. Nous proposons aux entreprises de financer les ateliers. Nous nous chargeons de la logistique (ordinateurs portables, reportage photo, clés USB pour emporter sa création à la maison, goûter, etc.). Les entreprises s'engagent à nos côtés pour offrir cet enseignement aux filles, pour qu'elles aussi découvrent cet univers passionnant qu'est l'apprentissage du <CODE>.



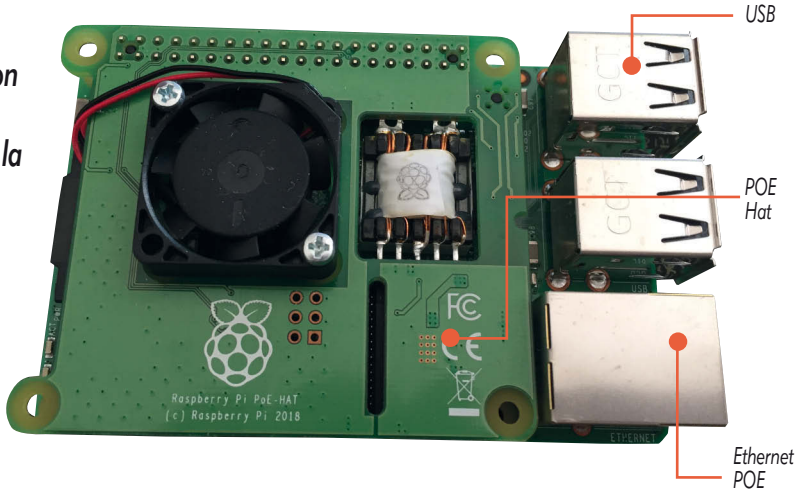
Évolution du nombre de filles dans une grande école d'ingénieur

Bonne nouvelle, la proportion d'étudiantes dans les écoles d'ingénieurs est en hausse depuis plusieurs années. À l'ESIEA (École supérieure d'informatique, électronique, automatique) par exemple, une des plus anciennes écoles d'informatique, les filles représentaient moins de 10% des effectifs dans les années 80s et 90s et jusqu'au milieu des années 2000. Ce chiffre avait grimpé jusqu'à 19% à la rentré 2012 mais avait chuté en dessous des 5% en 2015, pour finalement remonter à 22% cette année.

François Tonic

Le POE s'installe sur les cartes

L'alimentation sur le port Ethernet (ou POE ou Power over Ethernet) permet une grande souplesse d'utilisation et évite de multiplier les blocs d'alimentation. Sur Raspberry Pi, le POE se banalise, notamment avec la carte officielle : POE HAT.



L'idée du POE est très simple : alimenter un IoT, un périphérique, une carte via le port Ethernet. Le POE repose sur une norme : 802.3 xx. Généralement, on trouve sur les matériels POE la norme 802.3 af. L'alimentation délivrée va jusqu'à 15,4 W, au niveau du commutateur / injecteur. Au niveau du périphérique, elle plafonne à 12,9 W pour une tension de 48V. La norme 802.3 at, ou POE +, délivre jusqu'à 30W (et minimum 24W) pour une tension de 48V.

Comme nous sommes en Ethernet, le courant passe par le câble Ethernet. Cela signifie que les paquets de données et l'alimentation transitent sur le même câble, comme on peut l'avoir avec l'USB.

Les avantages du POE :

- Evite d'installer un câblage électrique ;
- Simplifie l'installation ;
- Simplification du déploiement des périphériques IP ;
- Economie dans l'installation et le matériel.

Il y a aussi des contraintes :

- Utiliser un câble cat. 6A, 7 ou 7A ;
- Déployer un routeur / injecteur POE ;
- Les périphériques doivent être compatible POE ;
- Ne pas dépasser 60 à 80 mètres de longueur de câble pour minimiser les pertes de signal ;
- Si vous utilisez un réseau 1Gb, votre routeur / injecteur POE doit être compatible pour éviter de créer un goulet et une perte de performance ;
- Faire attention à la consommation de vos périphériques / IoT, surtout si vous en avez plusieurs sur votre réseau POE. Si vous avez une consommation totale de 150 watts, votre routeur POE doit délivrer au minimum la même puissance.

La norme définit plusieurs classes :

- 0 et 3 : jusqu'à 15,4 w
- 1 : 4 w
- 4 : 30 w
- 5 : 45 w
- 6 : 60 w
- 7 : 75 w
- 8 : 90 w

Pourquoi un routeur ou un injecteur POE ?

Par défaut, Ethernet ne transporte que des données. Pour pouvoir mettre en place un POE, il faut donc un réseau Ethernet compatible. Plusieurs solutions s'offrent à vous :

- Installation du switch compatible POE : c'est souvent le plus simple. Si vous utilisez un réseau 1Gb, vérifiez bien que le switch soit aussi 1 Gb. Second point à vérifier : le nombre de ports POE car parfois, ils ne le sont pas tous.
- Passer par un injecteur POE : cela évite de remplacer votre switch. L'injecteur s'installe entre votre switch et le périphérique. Souvent – cher qu'un switch. Attention : l'injecteur nécessite d'être alimenté.

Bien entendu, un périphérique non POE ne devient pas POE par magie ! Chaque périphérique fournit sa classe et donc l'alimentation nécessaire à son fonctionnement. Le switch adapte alors celle-ci.

L'injecteur injecte dans le câble l'alimentation. Mais si vous utilisez une infrastructure POE avec des périphériques non POE, vous devrez utiliser un séparateur POE. Celui-ci va alors séparer la partie donnée et la partie alimentation. Switch ou injecteur ? L'inconvénient de l'injecteur est de rajouter des boîtiers supplémentaires avec un cordon d'alimentation pour chaque bloc, sauf si vous trouvez un injecteur multiports, mais à chaque port POE, vous devez le mapper à un port Ethernet du switch. Au final, le coût n'est pas forcément plus intéressant qu'un switch, surtout si vous devez en installer plusieurs.

POE HAT pour Raspberry Pi

La fondation Raspberry Pi a lancé il y a quelques mois sa propre carte POE pour les Pi 3B+ (et uniquement ce modèle). Il est compatible 802.3 af. Il supporte la classe 2 et fournit un voltage en entrée de 36 à 56 V et 5 V en sortie. Il possède un ventilateur pour refroidir la carte.

L'installation est très simple :

1 on place le POE HAT sur les GPIO, le logo framboise doit être visible ;
2 on fixe le HAT avec la visserie livrée ;
3 on connecte le câble ethernet au Pi 3.

Le constructeur conseille de mettre à jour le système. Le ventilateur s'active et se désactive automatiquement, selon la température du processeur. Aucune modification sur la Pi n'est nécessaire pour utiliser le POE. •

Samsung X5 est annoncé comme le SSD le plus rapide sur le marché. Il s'appuie sur la technologie ultra rapide NVMe en interne et une connectique Thunderbolt 3 en externe. Le constructeur annonce 2800 Mo/s en lecture et 2300 Mo/s en écriture. Aujourd'hui, un SSD monte jusqu'à 850 Mo/s. Disponibilité depuis septembre. Tarif encore inconnu. Mais ce type de SSD, en version 500 Mo, est vendu 500-600 €.

Wear OS : nouvelle version

Google a dévoilé la nouvelle version du système Wear OS pour les montres et les matériels vestimentaires. L'interface a été largement modifiée notamment pour les notifications, les interactions avec Google Assistant et l'intégration des services Google Fit. Fit a lui aussi été entièrement revue sur la partie interface. Cette version doit arriver courant septembre.

Des nouvelles du langage Go version 2

La v2 de Go avait été annoncée il y a un an. Fin août, les premiers drafts ont été annoncés. Plusieurs grosses avancées sont attendues : gestion des paquets revue, meilleure gestion des erreurs, apparition des génériques. Beaucoup de discussions ont été engagées avec les communautés pour définir les meilleures implémentations. Cette version devrait voir le jour courant 2019.

En attendant, Go 1.11 a été livré le 24 août. Là encore, pas mal de changements sont annoncés : toolchain, runtime, librairies, modules et WebAssembly. Les modules sont appelés à remplacer le Gopath en intégrant le versionning et la distribution de paquets. Mais pour le moment, le support est expérimental, donc à ne pas mettre en production. La v2 du langage devrait stabiliser cette fonctionnalité.

La v1.11 en détail : https://golang.org/doc/go1.11

Un nouveau langage pour .Net

Lizzie est un langage de script pour .Net basé sur le modèle de conception des "délégués symboliques" . Lizzie permet d'exécuter des scripts créés dynamiquement. Ces scripts ne sont ni compilés ni interprétés, mais ils "compilent" directement vers les délégués CLR managés.

Lizzie est fortement influencé et inspiré de Lisp, mais sans la "notation polonaise" non intuitive expliquent les concepteurs de ce nouveau langage. (comprendre : sans les parenthèses de Lisp qui souvent rebutent les développeurs qui n'y sont pas habitués ou qui n'écrivent pas leur code comme il se devrait). La nature dynamique de Lizzie vous permet d'exécuter des morceaux de code Lizzie en ligne dans votre code C #, en chargeant votre code depuis des fichiers, ou en récupérant par exemple le code d'une base de données ou même en transmettant du code sur le réseau.

Une caractéristique très intéressante de Lizzie est la possibilité de faire une liaison (binding) entre du code Lizzie et un type CLR.

Site : https://github.com/polterguy/lizzie

Baromètre HIRED recherche d'emploi :

cet été, la demande pour les développeurs DevOps a explosé

Le DevOps, ça s'en va et ça revient. Comme l'a déjà démontré le baromètre mensuel Hired de la recherche d'emploi des développeurs, la demande pour certaines technologies, souvent les moins pratiquées, peut fluctuer massivement suivant les périodes.

Et c'est donc le cas du DevOps. Pourtant mis en avant dans seulement 0,58 % des candidatures référencées sur Hired entre Juillet et Août cette technologie a suscité en moyenne 9,3 demandes d'entretiens auprès des candidats, soit presque deux fois plus qu'en juin. Ce mois-là, 0,85 % des demandeurs avaient mis le DevOps en avant. Avec 6,8, 5,9, 5,9 et 5,8 demandes d'entretiens suscitées, React, Go, Angular et Python complètent le top 5 des technologies les plus demandées par les recruteurs. Go est d'ailleurs le seul rescapé du top 5 de juin puisque iOS, Vue et .Net s'en sont fait éjecter. La demande pour iOS a d'ailleurs été divisée par deux cet été, passant d'une moyenne de 6 demandes d'entretiens suscitées en juin à 3,2 en juillet et août.

Du côté des candidatures, Java, Node et PHP sont toujours les technologies les plus mises en avant par les candidats et suscitent un engouement modéré des recruteurs. Elles ont suscité respectivement 4,6, 5,4 et 4,6 demandes d'entretiens au cours des deux mois d'été, traduisant un certain regain d'intérêt pour ces langages qui semblaient délaissés par les recruteurs, en juin. De ce trio de tête, seul Node sort toutefois son épingle du jeu sur les 6 derniers moi avec une moyenne de 6 demandes d'entretiens suscitées. Java et PHP restent en queue de pelotons avec, en moyenne, 5,2 et 5,4 demandes d'entretiens suscitées. iOS et Android restent les technologies les moins demandées sur cette période.

Juillet/août 2018

Technologies demandées	Pourcentage de candidatures développeurs	Technologies demandées	Nombre moyen de demandes d'entretiens
Java	18,72%	DevOps	9.3
Node	14.59%	React	6.8
PHP	13.23%	Go	5.9
Python	11.48%	Angular	5.9
React	9.73%	Python	5.8
Angular	11.87%	Vue	5.6
Android	5.25%	Node	5.4
.NET	4.09%	Java	4.6
Ruby	3.50%	PHP	4.6
Go	2.72%	Ruby	4.5
Vue	2.92%	Android	4
iOS	2.33%	iOS	3.2
DevOps	0.58%	.NET	2.8

De mars à aout 2018

Technologies demandées	Pourcentage de candidatures développeurs	Technologies demandées	Nombre moyen de demandes d'entretiens
Java	18.63%	React	7.2
Node	14.75%	DevOps	7
PHP	13.92%	Go	7.5
Python	12.02%	Python	6.3
React	9.96%	Node	6
Angular	11.25%	Ruby	6
Android	5.17%	Angular	5.9
.NET	4.49%	Vue	5.8
Ruby	3.19%	PHP	5.4
Go	2.59%	Java	5.2
Vue	2.36%	.NET	4.9
iOS	1.67%	iOS	4.2
DevOps	1.06%	Android	3.8

A propos de HIRED

HIRED.com est une plateforme technologique de recrutement qui attire et sélectionne les meilleurs profils techniques du marché afin de permettre aux entreprises de recruter des candidats qualifiés rapidement et efficacement. Chaque semaine, entre 80 et 100 nouveaux candidats de la communauté française HIRED en recherche active (développeurs, data scientists, designers, etc.) sont triés sur le volet grâce à des algorithmes et prêts à être contactés.

Étienne BAUDOUX
Autodidacte, passionné d'informatique et en particulier des technologies Microsoft depuis mes 14 ans, j'aime apprendre en mettant directement la main à la pâte, quitte à retenter plusieurs fois jusqu'à trouver la bonne solution par moi-même.
http://www.velersoftware.com/

BaZic : introduction à la création de son propre langage de programmation interprété

Les langages de programmations se multiplient. Au fil des années, les besoins, utilités, syntaxes, modes et promoteurs changent. Interprété, compilé, ou les deux, à chacun sa tasse de thé. Dans cet article, nous allons nous pencher sur la création de notre propre langage de programmation simple. Nous étudierons principalement l'aspect théorique supporté par des exemples liés à un projet personnel que j'ai réalisé récemment.

Prenons le temps de réfléchir

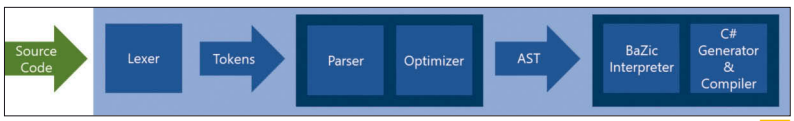
Se lancer dans la création d'un langage de programmation demande obligatoirement de prendre un temps de réflexion en amont pour trouver une ligne directrice à suivre. Posons-nous des questions… A quel besoin répond ce langage ou quel est son but et son utilité ? Compilé ou bien interprété ? Doit-il pouvoir fonctionner nativement ou bien sera-t-il dépendant de quelque chose ? Procédural ? Orienté-objet ? Haut niveau ? Bas niveau ? La liste est longue…

Pour les besoins de l'article, nous allons partir sur la création d'un langage inspiré du Visual Basic, procédural et interprété. « BaZic » sera un nom convenable (et pourquoi pas hein ?). Imaginons en exemple une syntaxe incomplète pour les besoins de l'article :

```
FUNCTION Fibonacci(n)
  # To return the first Fibonacci number
  IF n = 0 THEN
    RETURN 0
  END IF

  VARIABLE a = 0
  VARIABLE b = 1
  VARIABLE c = 0
  VARIABLE i = 2

  DO WHILE i <= n
    c = a + b
    a = b
    b = c
    i = i + 1
  LOOP
  RETURN END FUNCTION
```

Cette syntaxe doit être explicitée sous la forme d'une « grammaire » précise et stricte afin de couvrir tous les cas possibles de syntaxe du langage. Il existe plusieurs méthodes et outils pour créer votre grammaire. Vous pourrez trouver la grammaire complète du langage BaZic dans les crédits à la fin de cet article.

A présent, posons-nous de nouveau des questions : utiliserons-nous un outil ou framework pour automatiser la lecture du code source ? Ferons-nous une solution faite maison au maximum ? Sur quelle technologie déjà existante nous baserons-nous ?



1

Pour nous amuser au maximum, nous n'allons pas utiliser d'outils déjà existant pour simplifier la création et interprétation d'une grammaire. En revanche, je fais le choix personnel d'utiliser du .NET Framework pour me simplifier la gestion de la mémoire RAM et rester sur du haut niveau ; libre à vous de prendre n'importe quel autre langage et framework. Seule la difficulté changera.

Architecture

Les réponses à nos questions précédentes vont jouer sur l'architecture de notre « runtime » qui permettra de lire le code, de l'interpréter ou de le compiler. Souvent, un compilateur ou interpréteur pour un langage de programmation est composé de deux gros blocs :

1 • Lecture du code source et compréhension de sa structure.
2 • Interprétation ou génération du programme en fonction de la structure comprise.

Dans le cas du BaZic, ces deux blocs seront divisés en cinq parties. Trois pour la lecture et compréhension du code source, et deux pour l'interprétation. 1

1 • Lexer
Ce composant a pour but de convertir le code source textuel vers un tableau de jetons représentant tout ce qu'il y a dans le code (nous y reviendrons d'ici peu).

2 • Parser
Le Parser va ensuite utiliser ce tableau de jetons pour détecter des suites logiques afin de générer un arbre syntaxique abstrait (on détaillera plus bas).

3 • Optimizer
Durant cette étape, on va chercher à faire des ajustements dans l'arbre syntaxique abstrait pour optimiser en vitesse le code, ou faire tout un tas d'arrangements qui simplifient la vie à l'interpréteur.

4 • Interpreter
Ce composant va donner vie au code source en tentant d'exécuter des tâches en fonctions des opérations et expressions qui se trouvent dans l'arbre syntaxique abstrait.

5 • Translator & Compiler
Ce composant ne sera pas étudié dans cet article mais est disponible dans les sources à la fin de celui-ci. Il permet convertir le code BaZic en C# et de le compiler.

Lexer

Comme expliqué rapidement ci-dessus, la première étape est donc le Lexer. Son but est de prendre un code source, sous forme de texte, et de le découper mot par mot afin de générer un tableau de jetons qui représente le contenu du code source.
Mais au fait, qu'est-ce qu'on entend par « jeton » ici ?
Un jeton est un objet qui va contenir plusieurs informations :

1 • Le type de jeton
Est-ce que le bout de texte détecté dans le code source est un mot clé, un nombre entier, un symbole ?

2 • Le texte découpé
Le mot clé, nombre entier ou symbole tel qu'il apparaît dans le code source.

3 • La position et la longueur du jeton dans le texte.
Cette information sera utile dans le Parser.

Dans l'exemple explicité dans le schéma 2, on a découpé cette ligne de code en 8 jetons :

1 • Mot clé : VARIABLE
2 • Identifiant : « result »
3 • Opérateur : symbole égale « = »
4 • Parenthèse gauche
5 • Nombre entier : « 9 »
6 • Parenthèse droite
7 • Fin de la ligne

On peut constater que l'on ne tient pas compte des espaces et tabulations car, dans le cas spécifique de ce langage de programmation, ils ne sont pas importants. En revanche, on indique, via un jeton, où se trouve la fin d'une ligne de code, car dans ce langage, il y a une opération par ligne (comme en Basic, Visual Basic, Visual Basic.Net, etc.).

Parser

Une fois que tout le code source a été transformé en un tableau de jetons, celui-ci est envoyé au Parser. Ce composant va analyser l'ordre des jetons dans le tableau pour y détecter des suites logiques afin de déterminer les opérations et expressions dans le code. Ces opérations et expressions détectées vont faire partie d'un ensemble que l'on appelle « arbre syntaxique abstrait ». En anglais, Abstract Syntax Tree (AST).
Reprenons en exemple la ligne de code précédente.

```
VARIABLE result = Fibonacci(9)
```

Cette ligne représente la déclaration de la variable « result » ayant pour valeur par défaut le résultat d'un appel à la méthode Fibonacci de 9.
Dans le Parser, un ensemble d'algorithmes permettent de détecter la syntaxe exprimée par la grammaire (simplifiée) suivante :

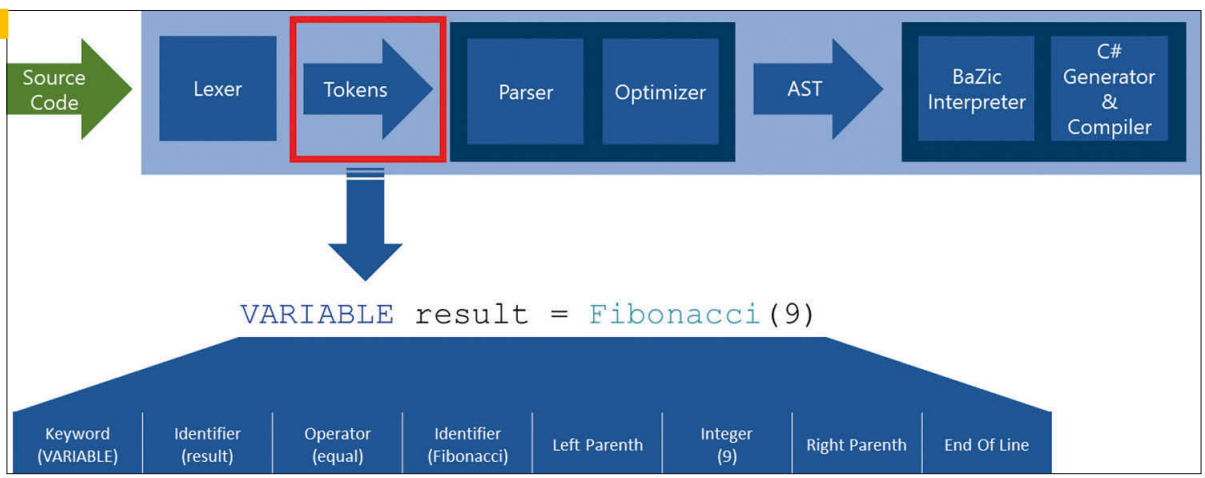
```
'VARIABLE' Identifier ('=' Expression)?
```

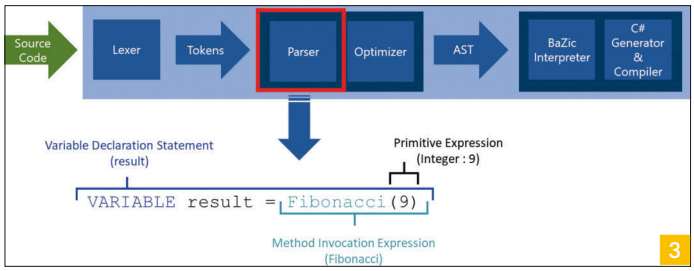
Comprenez ici, le mot clé « VARIABLE », suivi d'un identifiant, puis facultativement suivi du symbole d'égalité (=) et d'une expression.
Ce que fait le Parser exactement peut être exprimé dans le monologue suivant :

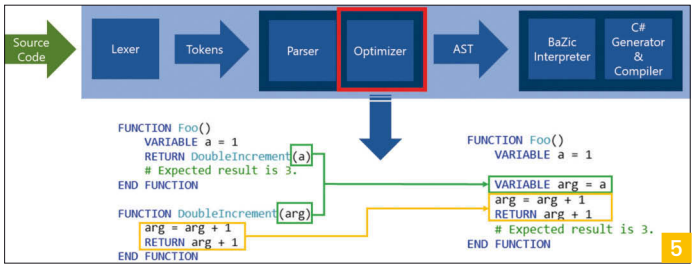
1 • Est-ce que la ligne commence strictement par le mot-clé « VARIABLE » ?
2 • Oui ? Super ! Qu'est-ce qu'il y a ensuite ? Un identifiant ?
3 • D'accord donc nous pouvons affirmer que cette ligne est la déclaration d'une variable ayant pour nom l'identifiant en question (dans notre exemple, « result » est l'identifiant).
4 • Ensuite, nous sommes autorisés à avoir un symbole égal (facultatif d'après la grammaire). On peut constater qu'il y en a un.
5 • Après un symbole égal, nous devons avoir obligatoirement une expression.
6 • [Pour simplifier, nous n'allons pas détailler ce qu'on fait pour analyser les expressions]
7 • Super, donc la valeur par défaut de la variable déclarée est l'expression en question. 3

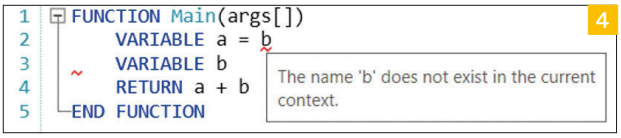
C'est ainsi que l'on peut générer l'AST (Abstract Syntax Tree, arbre syntaxique abstrait) qui représente notre programme codé en BaZic.
En allant un peu plus loin, on peut rajouter la condition suivante (non-exhaustive) :

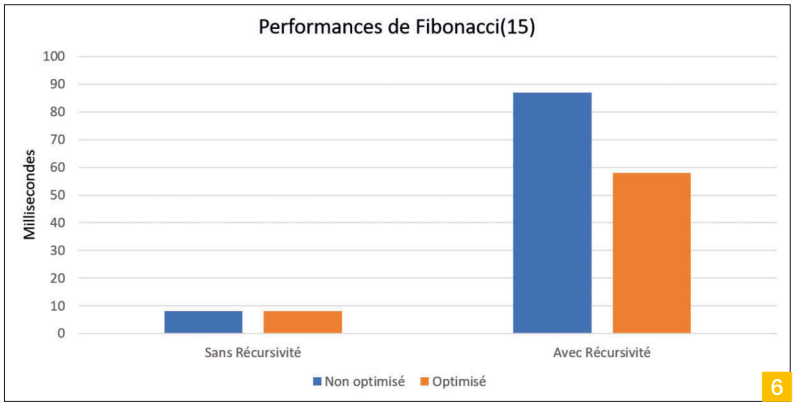
2











1 • Si la ligne commence strictement par le mot-clé « VARIABLE »,
2 • Et que le jeton suivant n'est pas un identifiant, alors on a une erreur de syntaxe.

Et c'est ainsi que l'on peut utiliser les informations sur la position du jeton dans le code source pour informer l'utilisateur qu'il a fait une faute. 4

Optimiser

Un langage interprété, d'autant plus fait avec un langage de haut niveau, est globalement lent. Vous vous en rendrez sûrement compte une fois l'interpréteur codé et/ou essayé. Plusieurs choses peuvent être faites pour améliorer sa vitesse, en particulier au niveau de l'interpréteur lui-même. Mais il faut penser en-dehors de la boîte.

Une autre possibilité est de modifier l'AST afin de l'améliorer en vitesse sans modifier son comportement. Faire du Refactoring automatiquement en d'autres termes. C'est là que le composant Optimizer est intéressant.

De nombreuses optimisations de l'AST sont possibles. Dans le cadre de cet article, nous allons étudier un cas en particulier : l'alignement des méthodes.

Dans beaucoup de langages informatiques, les appels aux méthodes sont des actions coûteuses en performance car la création d'un nouveau Scope requiert beaucoup de travail pour le CPU. Nous ne nous en rendons pas forcément compte car les runtimes de la plupart de nos langage favoris font déjà cette optimisation. Mais dans notre cas, c'est à nous de développer le runtime. Donc c'est à nous de faire cette optimisation.

Un autre problème lié aux appels de méthodes est le dépassement de pile d'appels (Call Stack Overflow). Comme l'interpréteur va exécuter les opérations et expressions de manières récursives, une StackOverflowException (ou autre similaire au langage que vous utilisez) arrive rapidement. L'alignement des méthodes dans l'AST peut aider à régler ce problème, car cela réduira le nombre d'appels récursifs et réduira donc la pile d'appels de l'interpréteur.

Dans le schéma 5, on a aligné la méthode « DoubleIncrement » à l'endroit où elle est appelée. Ainsi, au lieu d'appeler la méthode Foo, et dedans, d'appeler la méthode « DoubleIncrement », on n'appelle plus qu'une seule méthode.

Dans le cas de la suite de Fibonacci exécutée de manière récursive (qui est une manière lente de procéder dans le cas de Fibonacci), cette optimisation permet un gain de vitesse d'environ 33%. 6

L'AST final est ensuite envoyé à l'interpréteur.

Interpreter

Comme évoqué brièvement plus tôt, l'interpréteur va tenter d'exécuter de manière récursive les opérations et expressions de l'AST. Ce n'est pas très complexe en soit. Il y a juste beaucoup de cas à gérer. Cette partie concerne principalement une implémentation dans un langage de haut niveau.

Variables

Très simplement, pour chaque Scope créé, un dictionnaire est instancié en mémoire. On met dedans le nom de la variable et sa valeur associée.

Reflection

Dans les langages de haut niveau, la Reflection permet d'accéder dynamiquement à des propriétés et des méthodes non prévues au moment de la compilation de notre interpréteur. Néanmoins, utiliser la Reflection fournit des performances moindres. Dans l'interpréteur BaZic, un ensemble de classes a été fait pour optimiser les performances de la Reflection en utilisant un cache.

Sources

Envie d'essayer d'exécuter du BaZic ? Envie de découvrir ce qu'il y a sous le capot et comment ça fonctionne niveau code ? Envie de le réutiliser dans un projet ?

Vous pouvez trouver tous les codes sources liés à cet article sur GitHub : https://github.com/veler/BaZic •

Yvan PHELIZOT
Craftsman chez Arolla
(yvan.phelizot@arolla.fr)

Mon langage de programmation sans prise de tête

Partie 1

Ah, les langages de programmation ! Sans eux, un développeur ne pourrait pas faire grand-chose. Les langages permettent aux développeurs de dialoguer avec la machine. Finalement, on les utilise tous les jours sans trop savoir d'où ils viennent, ni comment ils fonctionnent. C'est qu'ils nous obligent à adopter un paradigme (procédural, objet, fonctionnel, …).

Pourquoi s'intéresser à ce sujet ? Beaucoup d'entre vous ont, sans doute, étudié la théorie des langages et ne l'ont jamais appliquée. Eh bien, voici l'occasion rêvée de mettre nos cours en pratique !

Dans cet article, nous donnerons des définitions de notions liées à la théorie du langage. Nous réfléchirons à comment procéder pour créer un langage simple et nous mettrons en oeuvre un interpréteur pour ce langage.

Euh, mais pourquoi ça devrait m'intéresser ?

Bonne question ! C'est vrai, après tout, un langage de programmation, c'est un truc de "chevelu". Malgré tout, il est bien de savoir comment tout cela fonctionne. Et pour vous en convaincre, je vais donner plusieurs raisons :

- Tout d'abord, par simple curiosité. Il faut savoir que jusque dans les années 1960, écrire un compilateur était une tâche ardue et complexe. La théorie des langages a permis de rendre cela plus simple ([**HISTORY**]).
- On peut vouloir étendre nos outils. Un analyseur statique s'appuie sur l'arbre syntaxique abstrait pour détecter les problèmes. Vous pouvez jeter un coup d'oeil à **[PMD]** qui sert de base à de nombreux outils d'analyse de code.
- Si vous êtes adepte de katas, vous connaissez peut-être le "RPN calculator". Le but est d'implémenter un calculateur en polonais inverse (par exemple, 1 2 + retourne 3). Si vous ne connaissez pas la théorie des langages, terminer ce kata peut se révéler une tâche ardue. Il est important de connaître les bonnes solutions pour ne pas réinventer la roue !
- Il existe de nombreux outils qui sont apparus afin de faciliter les échanges entre développeur et métier. Je pense notamment à Cucumber et Drools qui proposent une syntaxe très proche du langage naturel. Il peut vous arriver aussi de vouloir créer un DSL spécifique pour votre métier car vous vous sentez limité par les possibilités de ces outils.
- Si vous êtes adeptes de Natural Language Processing (NLP), il est important de connaître la théorie des langages et ce que l'on peut en faire ([**NLTK**]).

Il m'est d'ailleurs arrivé plusieurs fois de créer moi-même un langage pour répondre à un besoin particulier. Et en fait, ce n'est pas si compliqué !

Mais, dis-moi, c'est quoi un langage ?

On passe à la partie un peu plus formelle. Je ne présenterai pas la théorie des langages, juste les notions et concepts principaux. Il faut savoir que c'est Noam Chomsky qui a défini une algèbre pour pouvoir manipuler de manière formelle le langage.

Commençons par quelques définitions :

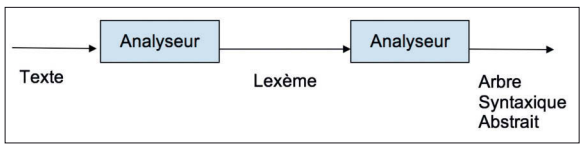
- On a tout d'abord un **alphabet** qui est un ensemble de symboles, que l'on appelle couramment des **lettres**. Par exemple : a, b, c, ...
- Une suite de lettres forme des **mots** (ou **lexème**). Par exemple, avion, bateau, …
- Un **langage** sur un alphabet est un ensemble de mots. Un langage peut être défini par une **grammaire** formelle (ou syntagmatique). Cette grammaire permet d'engendrer le langage. Une grammaire est un ensemble de règles décrivant les séquences de mots possibles.

Si on prend l'exemple du français, notre grammaire définit des règles comme :

S -> <SUJET> <VERBE> <COMPLEMENT>
SUJET -> <ARTICLE> <NOM>
ARTICLE -> un|une|le|la

Dans notre métier de développeur, un langage de programmation est un langage (avec ses mots et sa grammaire), dont le but est de décrire des traitements. Un programme n'est alors qu'un texte de ce langage.

Une fois que nous avons un langage de programmation (et sa grammaire), tout programme écrit dans ce langage peut être compilé ou interprété. En amont, on retrouve une phase d'analyse qui se décompose en deux parties : analyse lexicale qui transforme un groupe de lettres en mots, suivi de l'analyse syntaxique. L'analyse syntaxique construit généralement un arbre représentant la structure d'un texte en le dérivant à partir de la grammaire.



Vous connaissez Javascript ? Alors, vous avez déjà dû utiliser cette commande :

```
JSON.parse('{"a": 1, "b": "Hello"}');
```

Ici, le texte est **'{"a": 1, "b": "Hello"}'**, les lexèmes sont "{", string

("a"), ":", … Il ne reste plus qu'à vérifier si le texte respecte la grammaire JSON. C'est la partie analyse lexicale. Enfin, en dernière étape, **JSON.parse** transforme la sortie de l'analyse en un objet en mémoire représentant la chaîne de caractères.
Le parcours de cet arbre peut servir aussi bien à la compilation, qui traduit un programme d'un langage en un autre, qu'à l'interprétation, qui est l'exécution du programme. C'est à partir de cet arbre que nous tirons toute la force des langages. Il peut être interprété, utilisé pour transformer le code en un autre langage, optimisé, …
On parle généralement de **compilation** pour indiquer que le langage est transformé en langage machine et de **transpilation** pour un langage transformé en un autre langage.
Le design pattern "**Visitor**" vous a toujours semblé compliqué ? Sachez qu'il est d'une grande aide dès lors qu'on l'applique au traitement du langage.

Ok, j'ai compris. En gros, j'ai besoin d'une grammaire, c'est ça ?

Exactement. Les grammaires sont généralement décrites avec une syntaxe, comme la syntaxe Backus-Naur par exemple. Vous faites du Java? Eh bien, Java a aussi sa grammaire ([**JAVA-GRAMMAR**]).
Certains auront remarqué que les langages de programmation sont écrits dans la langue de Shakespeare. Soyons un peu puristes, écrivons notre langage dans la langue de Molière : le **frenchy**.
Nous allons créer un langage impératif simple avec les capacités suivantes :

- Type : entier ou booléen
- Variables
- Opérations : addition, égalité
- Structure de contrôle : if
- Fonctions

Voici un extrait de notre grammaire :

```
/*
* Les mots de notre langage
*/
BOOLEAN   : 'vrai'|'faux';
WORD      : ('a'..'z'+);
VALUE     : ('0'..'9'+);
WHITESPACE : ( '\t' | ' ' | '\r' | '\n'| '\u000C' )+ -> skip ;
ADD       : '+';
EQUALS    : 'égale';
/*
* Les règles de notre grammaire
*/
program          : (variableDefinition)+ EOF;
element          : VALUE|BOOLEAN|WORD;
operation        : EQUALS|ADD;
statement        : element (operation element)*;
variableDefinition  : 'soit' WORD 'valant' statement;
```

Il faut noter que l'on a un langage qui décrit un autre langage. Ce langage étant aussi représenté par une grammaire valide, on peut donc avoir un analyseur traitant le texte et qui produit en sortie un analyseur. On parle ainsi de compilateur de compilateur.

J'ai ma grammaire, c'est bon ! Maintenant, je fais quoi ?

Une fois qu'on a notre grammaire, on peut écrire un programme qui est capable d'interpréter des programmes écrits suivant cette grammaire. Heureusement, la théorie des langages est mature et a permis de fournir de nombreux algorithmes et programmes facilitant la génération des analyseurs lexicaux et syntaxiques.
Nous allons utiliser **ANTLR**. Comme son nom ne l'indique pas forcément, le rôle de ce framework est de faciliter la production d'un compilateur de compilateur, c'est-à-dire qu'il est capable de compiler une grammaire et d'en produire le compilateur associé. En sortie, le compilateur fournit l'arbre syntaxique tant recherché. L'intérêt de ce framework est notamment de permettre de générer du code dans de nombreux langages comme Java, C#, Python, Javascript, ...
Pour installer sous Ubuntu, il suffit de taper la commande suivante :

```
$ sudo apt install antlr
```

Il existe de nombreuses autres alternatives comme javacc, bison/flex, …
Le projet de démonstration a été fait en Java avec Maven ([**GITHUB**]). Le fichier de la grammaire doit se trouver dans **src/main/antlr4/<mon/package/destination>Frenchy.g4**.
Bien que la grammaire ait été présentée dans son état final, je conseille d'avoir une approche incrémentale pour pouvoir la construire petit à petit.

Mais, euh, si je me plante en écrivant un programme, qu'est ce qui se passe ?

Pas grand-chose. On peut regrouper les erreurs sous plusieurs types :

- Erreur au moment de l'analyse lexicale : ici, un mot n'est pas reconnu. L'erreur sera alors *"line 1:5 token recognition error at: '!'"*
- Erreur au moment de l'analyse syntaxique : on obtient une erreur du type *"line 1:0 mismatched input 'x' expecting 'soit'"* ou *"line 1:7 missing WORD at 'valant'"*
- Erreur au moment du traitement : c'est généralement là que les erreurs du type "variable inconnue" apparaissent. Il est nécessaire de parcourir le programme jusqu'à cette étape pour détecter une erreur de sens. Cette partie, c'est à nous de la gérer !

```
y = y + "2";
String y = 1;
```

D'abord, il y eu les types

Commençons par définir les types de bases :
Frenchy.g4

```
BOOLEAN   : 'vrai'|'faux';
VALUE     : ('0'..'9'+);
// Pour ne pas être sensible aux espaces
WHITESPACE : ( '\t' | ' ' | '\r' | '\n'| '\u000C' )+ -> skip ;
element          : VALUE|BOOLEAN;
program          : (element)+ EOF;
```

Le programme suivant en Frenchy suit notre grammaire :

```
123
true
```

Il nous faut maintenant générer le compilateur à l'aide d'ANTLR. Avec maven, c'est aussi simple que :

```
$ mvn clean generate-sources
[INFO] Scanning for projects...
...
[INFO] --- antlr4-maven-plugin:4.7:antlr4 (default) @ frenchy ---
[INFO] ANTLR 4: Processing source directory /home/yvan/Documents
/arolla/frenchy/src/main/antlr4
[INFO] Processing grammar: Frenchy.g4
...
```

Les fichiers suivants ont été générés dans **target/generated-sources** :

- **FrenchyLexer** : l'analyseur lexical.
- **FrenchyParser** : l'analyseur syntaxique.
- **FrenchyVisitor** : interface implémentant le pattern visitor.

Créons ensuite notre visiteur. Celui-ci affichera les éléments trouvés :

```
class MyFrenchyVisitor implements FrenchyVisitor<MyFrenchyVisitor> {
 @Override
  public MyFrenchyVisitor visitProgram(ProgramContext ctx) {
   System.out.println("value=" + ctx.VALUE());
   System.out.println("boolean=" + ctx.BOOLEAN());
  }
}
```

Et pour mettre tout ensemble :

```
// Récupérer une liste de symboles valides dans l'alphabet
String s = "<MON PROGRAMME>";
MyFrenchyVisitor monVisitor = new MyFrenchyVisitor();
FrenchyLexer lexer = new FrenchyLexer(CharStreams.fromString(s));
// Récupération des lexèmes
CommonTokenStream tokens = new CommonTokenStream(lexer);
FrenchyParser parser = new FrenchyParser(tokens);
ProgramContext program = parser.program();
program.accept(monVisitor);
```

Explique-moi ce qu'est un visiteur

Si vous êtes en grand amateur comme moi des design patterns, vous avez déjà dû entendre parler du pattern visitor.

Il fait partie de la classe des patrons de conception comportementaux. Il permet d'ajouter de nouvelles fonctions à une structure de données sans la modifier et sans exposer directement son état interne.

La classe visitée doit avoir une fonction **accept** pour recevoir le visiteur. La classe transmet les informations qu'elle contrôle au visiteur en appelant la fonction **visit** de ce dernier.

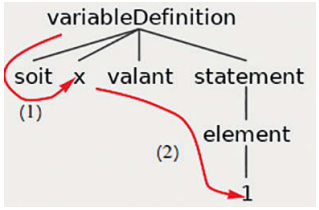
Puis les variables

Afin de pouvoir gérer les variables, il est d'abord nécessaire de les définir dans notre grammaire :

```
WORD    : ('a'..'z'+);
element        : VALUE|BOOLEAN|WORD;
variableDefinition  : 'soit' WORD 'valant' element;
program        : (variableDefinition)+ EOF;
```

Dans notre visiteur, nous allons introduire une structure pour contenir et gérer nos variables :

```
public Map<String, Variable> variablesByName = new HashMap<>();
```



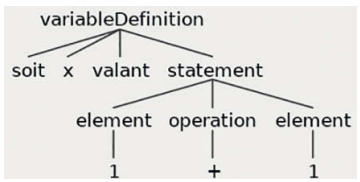
A l'étape (1), le programme détecte une nouvelle variable. A l'étape (2), la variable est initialisée et persistée.

Et ensuite les opérations

De même, nous modifions notre grammaire comme suit :

```
ADD    : '+';
EQUALS  : 'égale';

operation        : EQUALS|ADD;
statement        : element (operation element)*;
variableDefinition  : 'soit' WORD 'valant' statement;
```

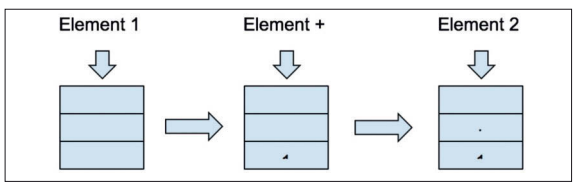
Le programme suivant devient donc :



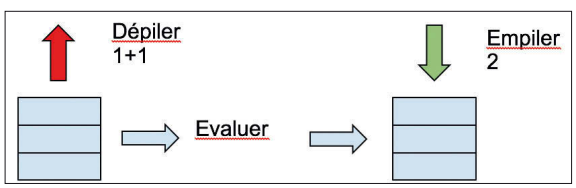
Ici, le problème se complique. Il va être nécessaire de connaître le résultat de l'addition avant l'assignement. Nous allons tout simplement évaluer l'expression et prendre la valeur pour l'assigner à notre variable.

Une solution est d'utiliser une pile à chaque fois que l'on rencontre un élément ou une opération, puis de dépiler les informations pour l'évaluer à la fin.

Lors du parcours de l'arbre, on remplit la pile :



Puis, on évalue les éléments à la fin :



Ainsi, le dernier élément restant est la valeur de la branche **statement** qui sera assignée à la variable.

Enfin, les structures de contrôle

Nous allons ajouter une structure de contrôle simple : if-then-else. Dans notre grammaire, nous trouverons donc :

```
ifStatement     : 'si' condition 'alors' statementThen 'sinon' statementElse;
```

Au niveau du visiteur, pour gérer la branche prise, il suffira d'éva-

luer la condition au moment du passage et de ne parcourir qu'une seule des branches :

```
if ((boolean) value.value()) {
  return ctx.statementThen().accept(this);
} else {
  return ctx.statementElse().accept(this);
}
```

Et pour terminer, les fonctions

Les fonctions suivent un peu le même principe que pour les variables. Nous aurons besoin d'une table pour stocker les noms des fonctions et leur corps. A chaque rencontre d'un appel à la fonction, l'arbre de traitement appelé sera exécuté.
Pour notre grammaire :

```
functionDefinition  :'étant donné la fonction' WORD 'retournant' functionStatement+ 'alors';
```

Et, au moment de l'exécution, pour éviter que le contexte de la fonction ne vienne écraser celle de notre programme, nous fournissons un nouveau contexte pour les variables locales.
Voici un exemple de programme que nous sommes capables d'interpréter à la fin :

```
étant donné la fonction f retournant
  si z égale 7 alors x + z sinon 9
  alors
étant donné la fonction g retournant
  soit y valant 1
  soit x valant z
  x + y
alors
soit x valant 5
soit z valant 7
soit w valant f + g
```

Je te crois pas, ce n'est pas comme ça "dans la vraie vie" ?

Et pourtant si ! Prenons l'exemple de OpenJDK, le jdk de java disponible en opensource. Sous Ubuntu, il est possible de télécharger OpenJDK 8 avec la commande suivante :

```
$ sudo apt-get install openjdk-8-source
$ cd /usr/lib/jvm/openjdk-8/
$ unzip src.zip
```

Par exemple, dans le dossier **com/sun/tools/javac/tree/JCTree.java**, on retrouve l'arbre de compilation de javac. Par exemple, le noeud pour représenter le "if" :

```
/**
 * An "if ( ) { } else { }" block
 */
public static class JCIf extends JCStatement implements IfTree {
  public JCExpression cond;
  public JCStatement thenpart;
  public JCStatement elsepart;
...
  @Override
  public <R,D> R accept(TreeVisitor<R,D> v, D d) {
    return v.visitIf(this, d);
  }
...
}
```

Et pour parcourir l'arbre, toujours notre célèbre pattern "visiteur". Si vous êtes intéressé par la partie arbre dans bytecode, regardez

```
/usr/lib/jvm/openjdk-8/com/sun/tools/javac/jvm/Gen.java
```

La suite de l'article le mois prochain

Ressources :

[GITHUB] Github du projet : https://github.com/cotonne/frenchy
[HISTORY] https://en.wikipedia.org/wiki/History_of_compiler_construction
[DANGLING_ELSE] https://fr.wikipedia.org/wiki/Dangling_else
[NLTK] http://www.nltk.org/book/ch08-extras.html
[PMD] https://pmd.github.io/
[JAVA-GRAMMAR] https://docs.oracle.com/javase/specs/jls/se7/html/jls-18.html
[HISTORY] http://rigaux.org/language-study/diagram.html

Damien Seguy
Directeur technique chez Exakat Ltd., spécialisée dans les solutions pour la qualité du code source en PHP. Il dirige le développement du moteur d'analyse statique Exakat, qui assure la revue de code pour les migrations, la clarté et la sécurité. Il aime faire du gremlin, des 狮子头 et du camembert.

Sur la piste de PHP 7.3

Depuis le 1er août, PHP 7.3 est passé en mode 'feature freeze' : gel de modifications. Toutes les nouveautés et modifications ont été estampillées, et il ne reste plus qu'à passer à la programmation et au débogage. Cela a déclenché la production de la version beta 1 le lendemain, et nous serons en version RC (Release candidate, candidat à la publication) dès la mi-septembre. Il est désormais grand temps d'ouvrir la boîte aux trésors, et de tester tout ce que PHP 7.3 a pour nous !

Modifications principales
– Amélioration du Garbage Collector ;
– Syntaxe Heredoc/Nowdoc modernisée ;
– Virgule terminale ;
– Obsolescence des constantes insensibles à la casse.

Infrastructure
– PCRE 2.0 ;
– SQLite 3.24.

Améliorations mineures
– json_encode() émet des exceptions ;
– Trouver la première et la dernière clé d'un tableau ;
– list() avec référence ;
– is_countable() ;
– net_get_interfaces() ;
– Chronomètre monotone : hrtime() ;
– compact() signale les variables non définies ;
– Interdiction de la fonction assert() ;
– Suppression de image2wbmp().

Amélioration du Garbage Collector

L'une des améliorations principales de PHP 7.3 est la refonte du Garbage Collector, aussi appelé GC.

Le garbage collector est un outil interne qui libère la mémoire. PHP accumule de nouveaux objets en RAM, et lorsqu'il atteint la fameuse limite de memory_limit, il fait appel à un nettoyeur : le garbage collector. Ce dernier passe en revue la mémoire utilisée, et tâche de recycler ce qui peut l'être. Une fois cela fait, PHP reprend le cours de son exécution.

La grande majorité des applications PHP n'ont jamais recours au garbage collector. D'abord, memory_limit est généralement assez élevé pour ne pas gêner le script, et il n'y a même pas besoin de recycler la mémoire : PHP libère tout à la fin de l'exécution. Ensuite, le garbage collector ne travaille que sur des racines : ce sont les structures dynamiques, comme les tableaux et les objets. Il faut alors en accumuler 10 000, littéralement, pour que le garbage collector entre en scène. Cette limite de 10 000 entre en jeu bien avant memory_limit.

Lorsque la limite de 10 000 est atteinte, le garbage collector est sollicité très souvent pour faire redescendre la liste des racines sous cette la limite. Il est notamment sollicité lors de la sortie d'un contexte d'exécution, c'est-à-dire une fonction, une méthode, une closure... Jusqu'à présent, toute application qui dépasse la limite des 10 000 voit le nombre d'appels au GC augmenter, et les performances décliner.

Avec PHP 7.3, le garbage collector est bien plus efficace. S'il ne peut libérer suffisamment de mémoire, il va simplement augmenter la limite de 10 000. Cela évite à PHP de faire trop d'appels inefficaces au recyclage, et cela garde le script performant. memory_limit, lui, n'est pas relevé et continue de s'exprimer, si besoin.

Les applications qui génèrent beaucoup d'objets, comme les applications en ligne de commande de longue durée, ou les applications à base de frameworks vont bénéficier de ces améliorations. Toutefois, la grande majorité des applications ne s'approchent pas de la limite du garbage collector, et ne verront pas de différence.

Pour aller plus loin sur ce sujet (en anglais) :
- *What About Garbage?*
- *How to optimize the PHP garbage collector usage to improve memory and performance?*
- *Improvements to Garbage Collection (GC) in PHP 7.3, 5x boost performance in tests*

Syntaxe Heredoc/Nowdoc modernisée

Heredoc et Nowdoc forment les troisième et quatrième formes de définition d'une chaîne de caractères en PHP. Elles sont pratiques pour les chaînes longues et multilignes :

```
<?php

$x = <<<FRANCAIS
Maître Corbeau, sur un arbre perché,
Tenait en son bec un fromage.
Maître Renard, par l'odeur alléché,
Lui tint à peu près ce langage:
Et bonjour, Monsieur du Corbeau,...
FRANCAIS;

?>
```

La syntaxe commence par un triple supérieur <, suivi d'un identifiant. Cet identifiant est un identifiant PHP classique : des lettres, des chiffres et des soulignés. L'identifiant est totalement libre : dans l'exemple ci-dessus, nous l'avons utilisé pour commenter la langue du texte. On peut s'en servir pour signaler du SQL, HTML, GREMLIN, XML, PHP, YAML, DOT, etc.

L'identifiant terminal est soumis à plusieurs contraintes : il doit être le seul sur sa ligne, et il ne doit partager cette ligne qu'avec un point-virgule. Il ne tolère même pas d'espace : dans ce cas, il bloque PHP avec l'erreur syntax error, unexpected end of file.

Bien entendu, le délimiteur final ne doit pas se trouver seul dans le

texte, sous peine d'être confondu avec le vrai délimiteur. C'est une situation exceptionnelle, ce qui la rend difficile à déboguer. Mais quand cela arrive, il suffit de rallonger l'identifiant.
NowDoc et HereDoc sont de proches cousins : Heredoc se comporte comme les chaînes à guillemets doubles. Il interprète automatiquement les variables, les tableaux à une dimension, et les propriétés simples. De son côté, Nowdoc entoure l'identifiant d'ouverture avec des guillemets simples, et se comporte comme une chaîne à guillemets simples : il n'y a aucun remplacement.
Dans les deux cas, les caractères spéciaux comme les guillemets simples et doubles n'ont pas besoin d'être protégés par des antislashs. Ces derniers, en revanche, ont toujours besoin de l'être pour ne pas être interprétés.

```
<?php

$animal = 'Corbeau';
// Heredoc
$x = <<<FRENCH
Maître $animal, sur un arbre perché,
FRENCH;

// Nowdoc
$x = <<<'FRENCH'
Maître $animal, sur un arbre perché,
FRENCH;

?>
```

En PHP 7.3, deux contraintes qui pèsent sur ces syntaxes ont été retirées : en premier lieu, le délimiteur final peut être librement suivi par d'autres opérateurs et espaces. Il est donc désormais possible de placer un Heredoc dans un appel de fonction.

```
<?php

$variable = 'THINGS';
print strtolower(<<<ENGLISH
ALL THOSE $variable
ENGLISH);

?>
```

Ensuite, le délimiteur final peut être séparé du début de ligne. Il n'y a pas besoin qu'il soit collé tout à gauche. Mieux encore, l'indentation du délimiteur final est automatiquement retirée au début de chaque ligne du texte défini. Par exemple :

```
<?php

function foo() {
    return <<<MESSAGE
    Voici le message
    MESSAGE;
}

print foo();
// affiche 'Returned Message'
?>
```

On voit que le message est indenté comme le délimiteur final. Mais cette indentation, bien pratique lors de l'édition, est retirée à l'exécution. PHP identifie les espaces devant le délimiteur final, et les retire.
Pour ne fâcher personne, les espaces ET les tabulations sont supportés pour l'indentation du délimiteur final : on apprécie la touche consensuelle. La règle est simple : choisissez un camp, et ne les mélangez pas. Désormais, Invalid indentation - tabs and spaces cannot be mixed est un message d'erreur PHP officiel.
Pour aller plus loin : Flexible Heredoc and Nowdoc Syntaxes

Virgule terminale

Une autre modernisation de PHP est la généralisation de la virgule terminale. Vous la connaissez déjà dans les définitions de tableaux, ou dans les listes d'espaces de noms groupés : elle est maintenant disponible pour tous les appels de méthodes et de fonctions.

```
<?php
$valeur = 'cookie';

setcookie("TestCookie", $value, time()+3600, "/~rasmus/", "example.com",);
setcookie("TestCookie",
          $valeur,
          time()+3600,
          "/~rasmus/",
          "example.com",
          1,
          );

?>
```

Le dernier argument, qui est en fait un void, ne sera pas envoyé à la fonction. Le but de cette évolution est de réduire la taille des deltas lors de l'archivage dans un système de contrôle de versions. Avec la virgule terminale, on peut ajouter un argument à la fonction sans créer un delta de deux lignes : la précédente, avec la nouvelle virgule, et la suivante.

```
  1,
```

Au lieu d'un delta de deux lignes :

```
  "example.com",    1
```

Notez que les définitions de fonctions et méthodes ne supportent pas cette fonctionnalité.
Pour aller plus loin : Allow a trailing comma in function calls

Obsolescence des constantes insensibles à la casse

Les constantes définies avec la fonction define peuvent être insensibles à la casse. Cette fonctionnalité est maintenant obsolète et

sera totalement supprimées en PHP 8.0.
null, true et false sont exemptées de cette évolution, et restent valides dans n'importe quelle configuration de casse. De même, les constantes magiques, telles que __function__ et __TRAIT__, même si ce n'est pas explicite dans la RFC.

```
<?php

// Ceci produit une erreur en PHP 7.3
define('FOO', 1, true);

echo FOO;

// Ceci produit une autre erreur en PHP 7.3
echo foo;

?>
```

La suppression des constantes multicasse devrait conduire à un code PHP plus propre : la pratique courante des constantes est de les définir et utiliser en majuscules.
D'un autre côté, cela réduit la complexité du moteur PHP, et élimine quelques bugs.
Dans un sondage rapide de 700 applications, nous avons trouvé 2% d'applications PHP qui définissent et utilisent des constantes insensibles à la casse. Cette fonctionnalité est rarement utilisée, mais pour ceux qui sont impactés, les modifications seront sérieuses.
Pour aller plus loin : Deprecate and Remove Case-Insensitive Constants.

PCRE 2.0

Les expressions rationnelles de PHP, aussi appelées regex, sont basées sur une bibliothèque externe : PCRE. PHP utilise cette bibliothèque depuis les temps anciens : elle est tellement utile qu'elle fait partie du coeur de PHP, et ne peut même plus être désactivée. Pourtant, PHP s'appuie toujours sur la version 1.0 de cette bibliothèque. Or, elle a été abandonnée et remplacée par une nouvelle version, PCRE2, publiée en 2015. Et cette nouvelle version est activement maintenue.
Le passage de PCRE 1 à 2 est un gros changement dans les appels à la bibliothèque : cela impacte uniquement le moteur de PHP, et Anatol Belski a fait un énorme travail d'intégration de la nouvelle bibliothèque.
Du point de vue de l'utilisateur, la nouvelle version fonctionne comme l'ancienne, à quelques exceptions rares. + L'option S est activée par défaut. PCRE applique désormais des optimisations supplémentaires. + L'option X est désactivée par défaut. Elle applique des validations de syntaxe supplémentaires. + Unicode 10 est utilisé, en remplacement de Unicode 7 pour PCRE1. Cela signifie que les regex supportent beaucoup plus d'emojis, de caractères et de langues. Les regex qui utilisent ou dépendent de l'Unicode seront donc impactées. + Certaines syntaxes invalides sont désormais refusées. C'est flou, mais visiblement, elles sont déjà exceptionnelles.
Comme PHP ne compile pas les regex lors de la phase de compilation de PHP lui-même, il est recommandé de faire un inventaire de toutes les regex de votre application, et de les tester avec PHP 7.3 et PCRE 2.0. Vous pouvez extraire un inventaire des regex avec des outils comme Exakat (voir par exemple celui de CodeIgniter). Puis en utilisant null comme paramètre, PHP effectue la compilation de la regex, rapporte les erreurs et retourne un false. Il ne reste plus qu'à diagnostiquer et corriger.

```
<?php
 if (false === @preg_match($regex, null)) {
    print "$regex is an invalid regex in pcre\n. Error : ".error_get_last()."\n";
 }
?>
```

PCRE 2.0 apporte de nouvelles fonctionnalités, qui ne sont pas encore explicitement exploitées en PHP. Le focus est actuellement fait sur la migration depuis PCRE 1.0, et, des nouveautés non définies à cette heure, arriveront dans de futures versions.
Pour aller plus loin : PCRE2 migration

SQLite 3.24

SQLite est une autre bibliothèque indépendante, intégrée à PHP. En PHP 7.3, elle supportera la version 3.24, qui a été publiée en juin 2018.
Cette nouvelle version ajoute le support des UPSERT. Les UPSERT sont inspirés de la commande du même nom en PostgreSQL. Un UPSERT est un INSERT, qui est suivi par une clause spéciale ON CONFLICT : si la ligne insérée existe déjà dans la table, la commande se transforme en UPDATE.
Cette commande est une proche cousine de la commande REPLACE, qui est déjà disponible en SQLITE. La différence est que UPDATE n'a pas besoin de modifier toutes les colonnes, alors que REPLACE doit fournir toutes les valeurs de toutes les colonnes avant l'exécution.
Voici un exemple, tiré de la documentation d'UPSERT :

```
CREATE TABLE vocabulary(word TEXT PRIMARY KEY, count INT DEFAULT 1);
INSERT INTO vocabulaire(word) VALUES('jovial')
 ON CONFLICT(word) DO UPDATE SET count=count+1;
```

Un mot est inséré dans la table vocabulaire et la colonne count garde le compte des tentatives d'insertion dans le dictionnaire.
Pour aller plus loin : version 3.24.

json_encode() émet des exceptions

PHP utilise deux fonctions pour traiter le code JSON : json_decode() et json_encode(). Malheureusement, elles retournent toutes les deux null lorsqu'une erreur survient lors du traitement des données. Et pourtant, null est un résultat valide lors du décodage JSON : par exemple, la chaîne "null" se décode en null (la constante insensible à la casse).
Lorsque décoder null fait partie du champ des possibles, il faut alors interroger la fonction json_last_error() pour s'assurer que null est valide ou que c'est une erreur.
PHP 7.3 introduit une nouvelle option pour ces deux fonctions : JSON_THROW_ON_ERROR. Cette option fait émettre aux fonc-

tions json_encode() et json_decode() une exception quand une erreur survient : elle peut alors être proprement interceptée avec une clause try/catch.

```
<?php

try {
    $read = json_decode($data, false, 512, JSON_THROW_ON_ERROR);
} catch (JsonException $e) {
  echo "les données entrantes ne sont pas valides\n";
}

?>
```

Le comportement par défaut reste le comportement traditionnel, afin d'assurer la compatibilité ascendante. Il est alors recommandé de faire un appel à json_last_error() après les appels à json_encode() et json_decode() pour tester les erreurs.
Pour aller plus loin : JSON_THROW_ON_ERROR.

array_first_key(), array_last_key()

Trouver la première et la dernière clé d'un tableau requiert de sérieux détours en PHP 7.2. Alors qu'il est facile de savoir quel est le premier élément dans un tableau à index autogénéré (réponse : c'est 0), c'est une autre paire de manches pour aller chercher la première clé d'un tableau.
Différentes solutions incluent :
Voir code complet sur www.programmez.com

Depuis PHP 7.3, il est possible d'utiliser la fonction array_first_key() pour accéder à cette valeur, sans déplacer le pointeur interne, ni lancer une boucle.
La même situation s'applique à array_last_key(), qui cible la dernière clé. Les solutions sont encore plus créatives que pour la première clé :

```
<?php

$array = ['a' = > 1, 'b' => 2, 'c' => 3];

// solution 1 :
reset($array);
end($array);
$key = key($array);

// solution 2 :
$key = array_keys($array)[count($array) - 1];

// solution 3 :
foreach($array as $key => $value) {
    // Trop souvent utilisée
}

?>
```

array_value_first() et array_value_last() faisaient aussi partie de la RFC, mais une fois que l'on a mis la main sur la clé, il est facile de trouver la valeur.

Pour aller plus loin : array_key_first(), array_key_last() and array_value_first(), array_value_last()

list() avec références

PHP a déjà la fonction list(), aussi connue sous la syntaxe [] (mais à gauche dans l'assignation...). Jusqu'à PHP 7.3, il n'était pas possible d'utiliser ces fonctions avec des références. C'est désormais le cas.

```
<?php
 $array = [1, 2];
 list($a, &$b) = $array;

 // et aussi bien sur

 [$a, &$b] = $array;
?>
```

Cette nouvelle syntaxe est équivalente à celle ci-dessous :

```
<?php
 $array = [1, 2];
 $a = $array[0];
 $b =& $array[1];
?>
```

Et, si ce n'est pas un problème que vous rencontrez souvent comme cela, vous pouvez aussi vous rappeler que list() est utilisé très agréablement avec foreach(). Par exemple, cette boucle va passer en revue le tableau, et assigner la valeur de 7 à l'index 'c'.

```
<?php

$array = [['c' => 1, 2], ['c' => 3, 4], ['c' => 5, 6]];
foreach ($array as list('c' => &$a, 1 => $b)) {
   $a = 7;
}
print_r($array);

?>
```

Pour aller plus loin : list() Reference Assignment

is_countable()

Cette nouvelle fonction a pour objectif de simplifier la vérification des valeurs qui peuvent être comptées. Comme la fonction count() émet des alertes lorsqu'on lui transmet de mauvaises données (Parameter must be an array or an object that implements Countable), is_countable() permet de les tester avant de les compter.
Is_countable() est en fait un simple raccourci : il remplace la condition suivante :

```
<?php

if (is_array($foo) || $foo instanceof Countable) {
  // $foo est comptable, ou \Countable
```

```
}

?>
```

Pour aller plus loin : is_countable

net_get_interfaces()

net_get_interfaces() est une autre nouveauté, qui liste toutes les interfaces réseau disponibles sur la machine hôte. Jusqu'à présent, il fallait utiliser php_exec() pour interroger le système, quand on en avait les autorisations. Désormais, net_get_interfaces() retourne les mêmes informations dans un tableau standard, et directement depuis PHP. Il a été porté sur tous les OS disponibles, y compris Windows.

Le résultat d'un appel à net_get_interfaces() ressemble à ceci :

Voir code complet sur www.programmez.com et gitHub

Pour aller plus loin : getting ip for eth0;

Suppression de image2wbmp()

image2wbmp() a été supprimé de l'API PHP. Il permettait de produire des images au format WBMP, et dispose d'un jumeau appelé imagewbmp(). Ce dernier est toujours disponible en PHP 7.3.

Cette fonction a été identifiée comme un doublon de imagewbmp().

Interdiction de la fonction assert()

assert() est une fonction PHP native. Elle appartient à l'espace de noms global, et vérifie si une affirmation est vraie ou pas : il suffit de traduire de l'anglais pour le comprendre. Il n'est donc pas possible de définir une fonction appelée assert, puisqu'elle existe déjà. Le problème se pose quand la fonction assert est créée dans un espace de noms, et que la fonction est appelée avec un nom court, c'est-à-dire par un simple assert($arguments).

Les appels à assert peuvent être désactivés totalement en désactivant les assertions avec la directive zend.assertions=0 ou via un appel à assert_options. Dans ce cas, PHP ignore simplement tous les appels à assert, y compris les appels à la fonction assert quand elle est dans un espace de noms. Un simple passage en production, où les assertions sont désactivées, engendre des problèmes de fonction assert inexistante.

Désormais, assert n'est plus une fonction, mais une structure de langage. Toutes ses occurrences sont donc réservées. Si vous y tenez, le mot assert est libre pour les noms de méthodes.

Pour aller plus loin : Deprecations for PHP 7.3

Continue pour les boucles, Break pour les Switch

Vous vous êtes déjà posé la question de la différence entre deux mots-clé de PHP : continue et break. Les deux contrôlent l'exécution, et ils sont souvent pris l'un pour l'autre. Et pourtant, ils ont théoriquement une utilisation distincte, comme l'explique Nikita Popov dans la RFC qui nous intéresse :

```
<?php

while ($foo) {
  switch ($bar) {
    case "baz":
      continue; // In PHP: se comporte comme "break;"
            // In C:  se comporte comme "continue 2;"
  }
}

?>
```

Continue et break se comportent de la même façon, alors que dans d'autres langages, comme le C, il y a une distinction entre les deux. PHP 7.3 émet désormais des alertes à la compilation : Drupal ou TCPDF.

Dès que continue peut être confondu avec break, PHP 7.3 émet cette erreur : "continue" targeting switch is equivalent to "break". Did you mean to use "continue 2"?.

Pour aller plus loin : Deprecate and remove continue targeting switch

Chronomètre monotone : hrtime()

Nous avons tous utilisé date() et time() pour nous donner la date et l'heure. Pour mesurer des durées, c'est-à-dire la quantité de temps entre deux moments, on utilise souvent microtime(true). microtime() retourne en fait la date et l'heure sous forme d'un nombre de secondes, et y ajoute les microsecondes, voire plus. Et pourtant, dès que l'on fait la différence entre deux moments identifiés par microtime(), un bug se tapit dans l'ombre.

microtime() se base sur l'horloge interne, et on suppose que cette horloge va toujours de l'avant : au moins, jusqu'en 2037... Pourtant, de nombreux événements peuvent modifier la progression de l'horloge interne : l'heure d'été et d'hiver, les secondes intercalaires (27 depuis 1972), la reconfiguration manuelle de l'horloge,...

PHP 7.3 présente hrtime() : c'est un chronomètre monotone. C'est-à-dire qu'il progresse toujours, sans jamais revenir en arrière. Il se comporte comme microtime() au niveau de l'API, et fournit une représentation ultra-précise de la date et l'heure. Il est destiné à être utilisé pour faire des différences, et non pas pour sa valeur absolue.

```
<?php

print_r(hrtime(true));
print PHP_EOL;
print_r(hrtime());

?>
```

Ceci a déjà été affiché, un jour :

```
828536158380710
Array
(
  [0] => 828536
  [1] => 158403415
)
```

hrtime() se présente comme une version moderne de microtime(). Toutes les mesures d'intervalles de temps doivent être faites avec hrtime(), et microtime() doit être réservé pour l'affichage de précision. En fait, hrtime() commence à compter à partir d'un certain point dans le passé.
D'un autre côté, hrtime() n'est pas affecté par les variations de l'horloge interne.
Pour aller plus loin : High resolution monotonic timer #2976 and Monotonic Clocks – the Right Way to Determine Elapsed Time

compact() signale les variables indéfinies

compact() est une fonction très pratique pour convertir une liste de variables en un tableau.

```
<?php

$foo = 'bar';

$array = compact('foo', 'foz');
// ['foo' => 'bar'];

?>
```

Jusqu'à présent, compact() ignorait sans un bruit les variables indéfinies. C'était au développeur de vérifier que le tableau était correctement construit, ou bien de l'envoyer à la fonction suivante sans vérification. Ces jours sont révolus.
Compact() est très utilisé par les systèmes de templates, pour manipuler un nombre arbitraire de variables.
Cette nouvelle fonctionnalité pourrait produire beaucoup d'alertes : PHP affiche une notice. Pour s'assurer de l'impact, il faut relire les logs. Au pire, vous pouvez retourner au comportement précédent en utilisant l'opérateur @.
Pour aller plus loin : Make compact function reports undefined passed variables

Migration vers PHP 7.3

Avec toutes ces nouveautés et incompatibilités, comment se préparer pour une nouvelle version de PHP ? Voici un résumé des fonctionnalités et leur impact sur votre code. 1
Il y a trois moyens pour préparer votre code : changer ce qui ne fonctionnera plus, identifier les portions de code qui profiteront des améliorations, ou simplement, attendre PHP 7.3.

Se préparer à PHP 7.3

Se préparer consiste à éliminer dès maintenant les incompatibilités du code courant et les remplacer par du code qui fonctionnera dans les deux versions. C'est le cas pour les expressions rationnelles de PCRE2 : il faut alors collecter les expressions rationnelles, les tester avec le moteur de regex de PHP 7.3, puis appliquer les corrections nécessaires.
D'un autre côté, il n'y a pas grand-chose à faire pour se préparer à SQLite 3.24, puisque l'évolution principale est une fonctionnalité nouvelle (UPSERT).
Les recommandations sont des points du code qui pourront profiter de la nouvelle version, par exemple is_countable(). Pour cela, il faut attendre la migration vers PHP 7.3 pour pouvoir l'utiliser, ou bien adopter une bibliothèque de compatibilité pour émuler la fonction en PHP 7.2.

Migrer puis revenir en arrière

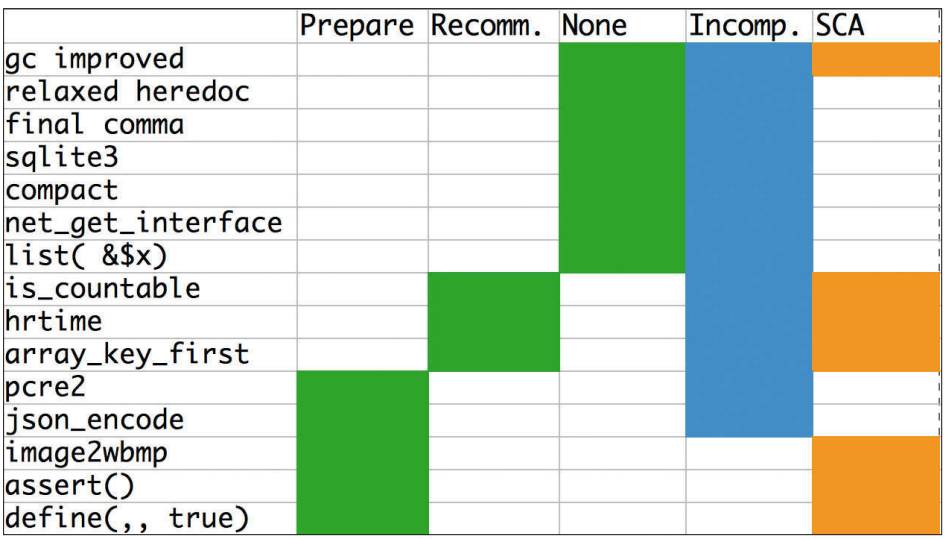
Finalement, le tableau signale les nouvelles fonctionnalités qui ne sont pas compatibles avec les anciennes versions. Une fois la migration faite, vous pourrez adopter les nouveautés de PHP 7.3, consciemment comme la virgule terminale, ou involontairement, comme le garbage collector amélioré. Toutes ces fonctionnalités rendent le retour en arrière impossible, car elles pourraient ne pas compiler dans les versions antérieures, ou simplement manquer cruellement. Une fois que vous aurez assuré la migration, pensez à bien évaluer si vous devez assurer la compatibilité ascendante.

Auditez votre code avant de migrer

La dernière colonne indique si l'analyse statique peut vous aider à relire et auditer le code. Des outils tels qu'Exakat fonctionnent déjà avec PHP 7.3-dev, et diagnostiquent efficacement les problèmes de migration. La relecture va bien au-delà de ce que PHP lint est capable de faire. Préparer la migration est le meilleur moment pour adopter ces outils et améliorer son code un peu tous les jours.

En attendant décembre

PHP 7.3 est prévu pour le 13 décembre. D'ici là, il est important de garder son code prêt pour la nouvelle version. Vous avez aussi votre rôle à jouer, à votre niveau. Voici quelques suggestions : + téléchargez PHP 7.3 depuis github et compilez-le + lintez votre code avec php -l, pour chasser les erreurs de syntaxe + Utilisez des outils d'analyse statiques tels que Exakat, pour passer en revue toutes les possibilités + exécutez votre suite de tests + signalez les bugs sur PHP Bugs •



1

Adrien PAVIE
Contributeur OpenStreetMap et entrepreneur en géomatique, je développe des solutions d'analyse et de visualisation de données géographiques pour mieux comprendre le monde qui nous entoure.
https://pavie.info/

OpenStreetMap : à la découverte de la base de données géographique libre ! Partie 1

niveau 100

Si je vous parle de cartes ou de plans, à quoi pensez-vous ? Vous imaginez sûrement la carte routière papier dans votre voiture, ou la carte de l'application GPS que vous avez dans votre téléphone. Une carte est une représentation du territoire, réalisée pour une finalité précise : se déplacer, recenser des objets, planifier des aménagements. Mais savez-vous comment on réalise ces cartes ? Comment passe-t-on de la page blanche à une carte riche en informations ? Non, ce n'est pas avec des crayons de couleurs comme vous le faisiez sur les bancs de l'école pendant vos cours de géographie !

Exemple de donnees vectorielles sur un campus universitaire

Les cartes sont depuis plusieurs décennies réalisées à partir d'informations brutes que l'on appelle données géographiques ou géospatiales. Ces données sont une abstraction de la réalité, et peuvent être collectées par différentes méthodes : campagnes terrain (des personnes vont sur place récolter des informations), interprétation d'images aériennes ou vues satellites (que ce soit automatiquement ou manuellement), et enfin collecte de données géolocalisées (localisation des véhicules de flotte d'entreprises par exemple). Le plus souvent, on stocke ces informations sous forme vectorielle, c'est-à-dire des géométries dont les nœuds sont localisés à l'aide de coordonnées spatiales. Ces géométries vont également comporter une description permettant de qualifier l'objet (les « attributs »). Quel est l'intérêt de disposer d'informations vectorielles brutes plutôt qu'une simple carte ? Posez-vous les questions suivantes : dans quelle mesure avez-vous la main sur les couleurs d'une carte (que ce soit papier ou numérique) ? Pouvez-vous facilement isoler les boulangeries d'une carte pour en créer une autre ? Non, car les cartes sont une représentation graphique des données brutes, potentiellement biaisée, sur laquelle vous n'avez pas la main. C'est un élément de communication, là où les usages les plus intéressants nécessitent de s'appuyer sur le matériau de base qu'est la donnée. Voyons les possibles lorsque l'on dispose d'une base de données géographique.

Des données, pour quoi faire ?

Lorsque l'on dispose des données brutes, les cas d'usages sont très nombreux. Tout d'abord, on peut créer des cartes : c'est certes l'usage le plus basique, mais également le plus courant. L'intérêt ici est que vous êtes maître de la représentation : choix des couleurs, des objets à représenter, de la zone d'étude, des polices d'écritures… En résumé, vous avez la main sur le style de la carte. D'ailleurs, les géants du web l'ont bien compris : ils ont le choix de ce qui apparaît sur leurs plans en ligne à grande audience, et y figurer a un prix. De la même manière qu'un e-commerce paie pour avoir une meilleure visibilité dans les résultats de recherche, un commerce physique paiera pour être mis en avant sur la carte. Maîtriser la représentation des données géographiques est donc essentiel.

Les données ont aussi un intérêt pour proposer des services : calculs d'itinéraires et recherches de lieux sont les plus communs. Imaginez que vous souhaitiez lancer une solution de guidage audio pour les personnes malvoyantes, en évitant les obstacles autant que possible (zones de travaux, escaliers). Très peu de services existent à l'heure actuelle, ce sera donc à vous d'interpréter l'information géographique, siqui elle existe, pour proposer des itinéraires. Ici encore, disposer des données permet donc d'avoir la main sur le traitement réalisé, et de personnaliser l'expérience utilisateur selon le public ciblé.

Si l'on prend maintenant du recul face aux applications concrètes, on peut également mener des analyses de plus haut niveau. Avec une base géographique, on est en mesure de réaliser des calculs d'isochrones (zones accessibles en un certain temps), rechercher des lieux d'implantations (quel sera l'emplacement idéal pour ouvrir mon commerce ou construire un nouveau lycée), ou découvrir des relations spatiales entre objets et phénomènes (vérifier la corrélation entre la présence d'équipements sportifs et le bon état de santé de la population). Ces études sont impossibles sans informations de qualité sur lesquelles s'appuyer.

Maintenant que nous avons une vision plus claire des possibles avec de bonnes données géographiques, voyons quelles sont les bases disponibles.

Comparatif des bases de données géographiques

Plusieurs structures proposent des jeux de données géographiques, plus ou moins fournis, de qualités et tarifs variables. Voici une vue d'ensemble des solutions proposées. (Voir tableau)

Pas évident de choisir ? Tout dépend de l'usage que vous voulez faire de ces données. Certains se posent peut-être la question : mais où se trouvent Google, Apple, Microsoft, Here ? En effet ces acteurs proposent des cartes en lignes grand public, mais ne sont pas créateurs des données vectorielles qu'ils utilisent (ou à la marge). Ces sociétés reposent sur les données fournies par les structures listées précédemment, même si elles créent des informations complémentaires (prises de vues 360°, recensement de commerces).

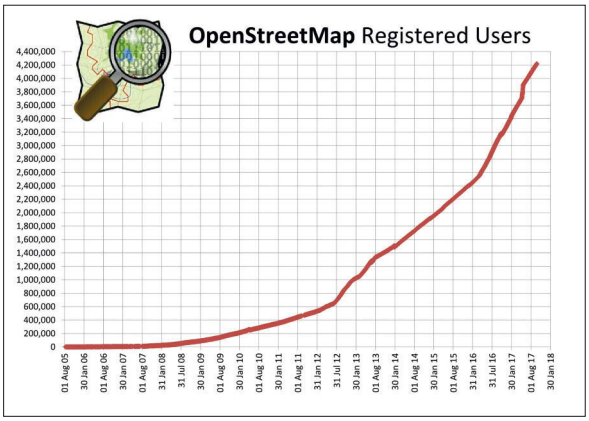
Si vous êtes à la recherche d'une base de qualité, à l'échelle mondiale, et qui plus est libre de droits, la meilleure option reste OpenStreetMap.

	Structures publiques	Entreprises	Projets collaboratifs
Exemples	Collectivités : villes, métropoles, régions Agences cartographiques : IGN (France), Ordnance Survey (Royaume-Uni)	TomTom, Navteq, DigitalGlobe, TeleAtlas	OpenStreetMap
Données vectorielles communément proposées	Réseau routier Bâtiments Équipements publics Démographie Occupation des sols	Réseau routier Info trafic	Réseau routier Bâtiments Équipements Commerces Occupation des sols Réseaux techniques Nature/environnement
Couverture	Territoire de la structure (ville, région, pays)	Mondiale	Mondiale
Exhaustivité	Très bonne	Bonne	Dépend de la présence de contributeurs Excellente en ville Variable en zone rurale
Richesse attributaire Périodicité	Moyenne Annuelle	Moyenne Hebdomadaire ou mensuelle	Excellente Base mise à jour en direct Relevés terrains selon présence de contributeurs
Politique tarifaire	Gratuit ou libre de droits	Payant et propriétaire	Gratuit et libre de droits

Le projet OpenStreetMap

Vous l'aurez compris, OpenStreetMap est une base de données géographique. Ce qui caractérise ce projet, c'est son côté ouvert et collaboratif : tout le monde peut participer et améliorer les données ! On le compare souvent à Wikipédia pour ce qui est du ressort de la cartographie, car il repose sur le même esprit de bien commun numérique. L'idée est de mettre en commun les efforts pour produire une base cartographique de référence, autogérée, qui soit utilisée par le plus grand nombre. Les informations sont renseignées principalement par des personnes bénévoles (mais pas uniquement), et la somme de ces contributions permet d'atteindre un bon niveau de qualité à l'échelle mondiale. Car qui connaît mieux un quartier que ses habitants ?

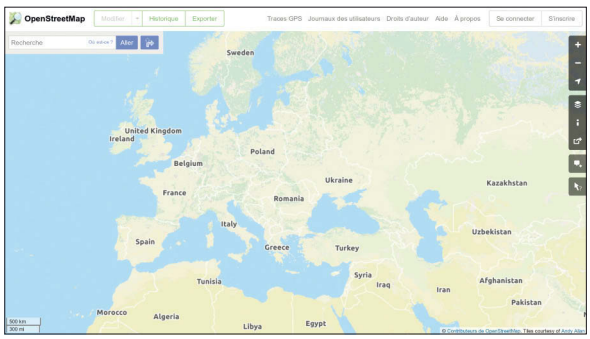
Le projet a été lancé en 2004 par Steve Coast, étudiant britannique à l'époque. Il part du constat que les agences cartographiques nationales (comme l'Ordnance Survey au Royaume-Uni ou l'IGN en France) disposent de données géographiques conséquentes, mais que leur mise à disposition est payante, alors que ces structures publiques sont financées par le contribuable. Pour contrer ce système, et partir dans la même logique vertueuse que Wikipédia, il lance OpenStreetMap. Au fur et à mesure des années, le projet est devenu mondial, avec une communauté de contributeurs en constante croissance, des partenariats noués avec d'autres acteurs pour utiliser leurs données (images aériennes Yahoo, Bing ou IGN, données ouvertes par les collectivités, informations terrain collectées par des entreprises…). En 2018, on recense plus d'un million de contributeurs différents à la base, et 90 millions de modifications en moyenne chaque mois. Les informations à disposition dans cette base sont très riches : d'une part, on trouve des objets de types très variés, que ce soit le réseau routier, les bâtiments, les commerces, les services, les occupations du sol, la végétation… D'autre part, les objets présents sont souvent qualifiés de manière très précise : on connaîtra pour une route son nom (et ses déclinaisons en langues régionales), le nombre de voies, la vitesse limite de circulation, le type de revêtement, la présence de bandes cyclables. Pour un bâtiment on disposera de son usage (résidentiel, commercial, industriel…), sa hauteur, son nombre d'étages, les matériaux utilisés, la forme de sa toiture. Bien sûr, la disponibilité de ces détails dépend du travail fourni par les contributeurs sur une zone. L'avantage, c'est que si les données viennent à manquer, il vous est toujours



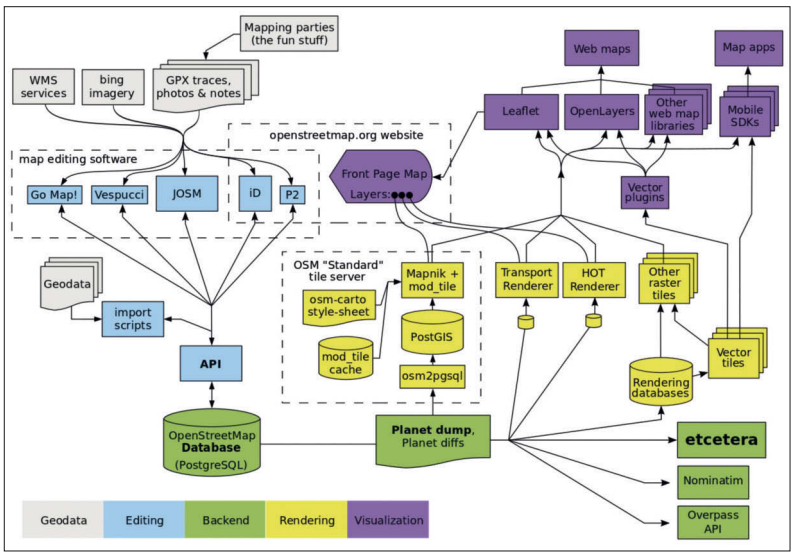
Nombre d'utilisateurs enregistres sur OpenStreetMap

Cles les plus utilisees sur les objets

Repartition et desserte des serveurs de mise a disposition du fond de carte



Page principale du site OpenStreetMap



Architecture logicielle d'OpenStreetMap

possible de les créer !

Comme pour Wikipédia, le projet est complètement ouvert aux contributions (à condition de se créer un compte, une formalité qui prend moins d'une minute). Une question récurrente est celle du vandalisme : y a-t-il des personnes malveillantes qui viennent dégrader les données, et quelles sont les mesures mises en place pour éviter cela ? Il faut savoir que le vandalisme « pur » représente une quantité presque négligeable des contributions. C'est peu courant, car pendant longtemps éditer OpenStreetMap nécessitait un temps d'apprentissage. Il fallait donc être particulièrement déterminé pour casser les données, ce qui décourageait la plupart des vandales. Second point, la visibilité du vandalisme est moins grande en comparaison que celle qu'on peut avoir sur Wikipédia. Là où l'encyclopédie est basée sur un mode texte avec des pages potentiellement polémiques, qu'il est donc facile de détourner, vandaliser une carte est plus subtil : faire passer une autoroute au milieu de la campagne, ajouter une insulte dans un nom de lieu touristique… Des modifications qu'il est facile de repérer automatiquement, qui n'a de visibilité que si l'on consulte la zone en question. Ces deux aspects limitent donc largement la casse intentionnelle. On retrouve plus fréquemment des problèmes introduits involontairement du fait du non-respect du formalisme lié s à la description des objets. Ces erreurs sont commises par des débutants bienveillants mais qui ne connaissent pas encore les subtilités techniques du projet. Dans ce cas, la communauté les accompagne afin qu'ils soient autonomes pour leurs prochaines contributions. Les contributeurs plus habitués disposent ainsi d'outils de suivi des modifications réalisées. Cela permet donc de vérifier les éditions des données, et d'en valider la bonne qualité. En cas d'anomalie, une discussion est engagée avec l'auteur à l'origine de la modification. S'il est de bonne foi, on l'aide à corriger l'erreur. S'il est visiblement mal intentionné, les changements sont annulés, et un système de blocage de l'utilisateur existe en cas de récidive. Ces mécanismes d'autorégulation de la communauté ont fait leurs preuves au fil des années, ce qui assure qu'OpenStreetMap propose des données de bonne qualité.

La qualité reconnue de cette base en fait une source de données incontournable. Elle est utilisée par des acteurs variés au niveau mondial : associations humanitaires (Croix Rouge américaine, Médecins sans Frontières), structures publiques (ministères, gendarmerie nationale, collectivités) ou privées (Facebook, Foursquare, divers journaux nationaux), et le grand public (à travers différentes applications en ligne ou sur téléphone). Maintenant que vous avez une vision plus claire de ce qu'est ce projet, et en quoi il est pertinent pour de nombreux cas d'usages, voyons à quoi ressemblent les données.

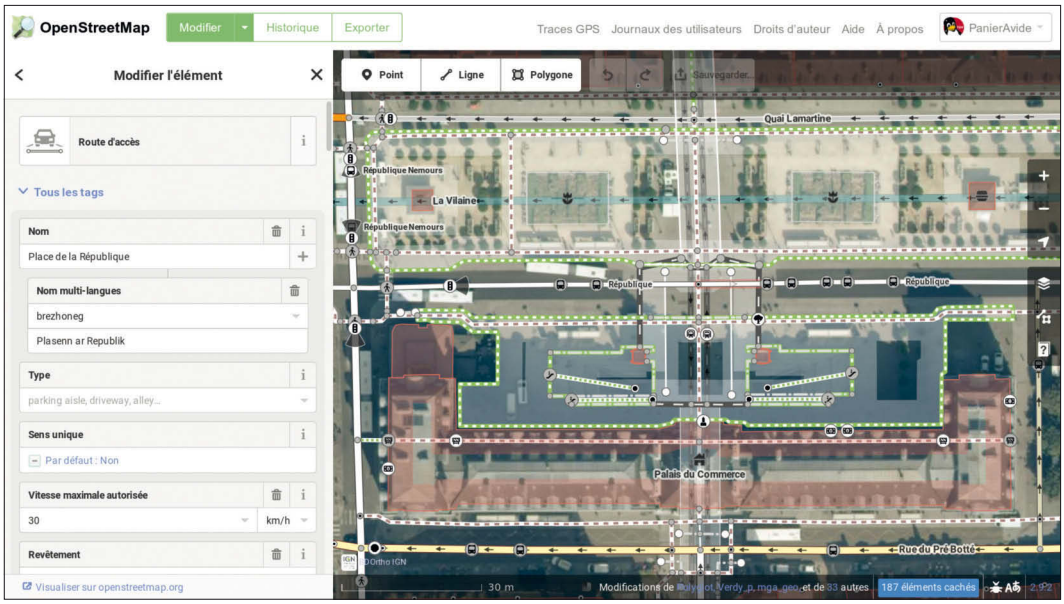
Structuration des données OpenStreetMap

De manière assez classique pour une base de données géographique, OpenStreetMap se compose de géométries, c'est-à-dire le dessin des emprises au sol des objets à représenter. Selon la taille et la morphologie de l'objet, on choisira de le dessiner de différentes manières :

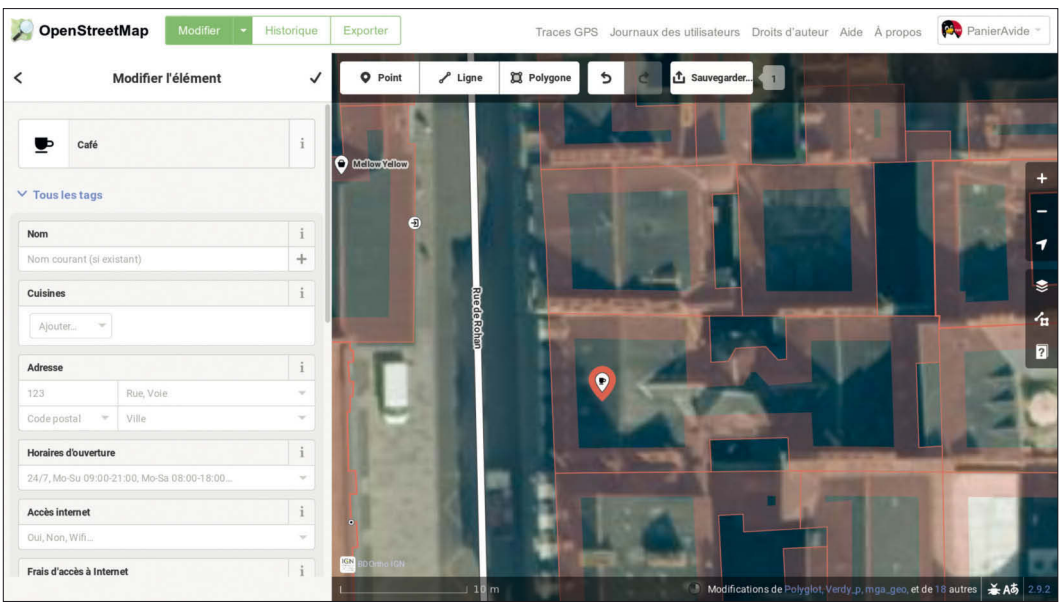
- Sous forme de nœud : un seul point géolocalisé.
- Sous forme de chemin (« way » en anglais) : une succession ordonnée de nœuds reliés entre eux. On peut également dessiner une emprise si le dernier nœud est identique au premier (par exemple pour représenter un bâtiment).
- À l'aide d'une relation : un ensemble d'objets ayant un lien entre eux, par exemple la succession des rues parcourues par une ligne de bus. Ce type d'éléments est assez complexe à manipuler et nécessite un peu de pratique avant de s'y intéresser.

À chaque géométrie est associée une liste d'attributs sous forme de clés et valeurs (« tags » en anglais). Ces associations de clés et valeurs permettent de catégoriser les objets. Exemple pour un restaurant :

• amenity = restaurant → il s'agit d'un lieu de restauration ;
- name = Le Bon Resto il porte ce nom précis ;
- cuisine = italian on y sert des plats italiens.
Il existe de très nombreux attributs que l'on peut renseigner : catégorie d'objet (bâtiment, commerce, équipement, loisirs…), nommage (nom actuel, ancien, local, traduits dans un ensemble de langues), accessibilité (par type de véhicule, de public, horaires) et autres détails spécifiques (type d'abri bus, espèce d'arbre, dimensions…). Il est à savoir que la liste des attributs est complètement ouverte : libre à vous d'en créer des nouveaux. Cependant, pour s'assurer de la bonne lisibilité des objets que vous ajoutez, il vaut mieux s'en tenir aux attributs existants. Ceux-ci couvrent les cas les plus courants, et il est plutôt rare de devoir créer une nouvelle clé (sauf à renseigner des objets métiers très particuliers). Vous retrouverez sur le wiki d'OpenStreetMap la liste des tags existants et la manière de les utiliser (le wiki est la référence documentaire du projet).
Maintenant que nous avons une vision claire de ce à quoi ressemblent les objets gérés par OpenStreetMap, voyons en détail son infrastructure.



Editeur iD.



Ajouter et decrire un café

L'infrastructure du projet

Pour la gestion de ses données, OpenStreetMap repose sur une API HTTP REST classique. L'API et le site web sont basés sur Ruby on Rails et utilisent le SGBD PostgreSQL. L'API récupère et envoie les objets au format XML en suivant un schéma qui lui est propre. En pratique, on communique avec l'API d'OpenStreetMap en direct lorsque l'on souhaite en modifier les données. Si l'on souhaite uniquement récupérer des objets, on passera par d'autres services dédiés, ce qui permet d'éviter des pics de charge en lecture.
Les serveurs principaux d'OpenStreetMap sont localisés au sein d'une université londonienne. Ceux-ci hébergent les services essentiels : API et site web de production. Les données sont répliquées régulièrement sur différentes machines à travers le monde. Des structures diverses mettent à disposition de la communauté des serveurs pour améliorer la continuité de service autour d'OpenStreetMap : système de cache, génération de fonds de carte, API d'interrogation de données… Cette infrastructure plutôt décentralisée a fait ses preuves et permet une disponibilité continue des services OSM. Par ailleurs, si vous souhaitez apporter votre soutien à l'écosystème, n'hésitez pas à contacter la communauté de bénévoles OpenStreetMap dans votre région : la communauté est ouverte aux partenariats.
Au-delà de l'infrastructure propre au projet OpenStreetMap, de nombreux acteurs sur le marché ont saisi l'intérêt de s'appuyer sur ces données pour proposer des services commerciaux à leurs clients. Ces prestataires permettent ainsi des utilisations en contexte de production industrielle (applications avec des milliers ou millions d'utilisateurs) sans perturber le fonctionnement de l'infrastructure cœur d'OpenStreet Map. Parmi les offres communes, on trouve la mise à disposition de fonds de carte (que ce soit pour de gros volumes ou avec une facilité d'édition du style), la recherche de lieux (ce que l'on appelle géocodage : transformer une adresse texte en des coordonnées géographiques), les calculs d'itinéraires (sur de gros volumes ou avec des profils spécifiques), et, enfin, des offres applicatives embarquant l'ensemble de ces services pour des développements sur mo-

bile. En plus de ces offres à destinations de grands comptes, on retrouve également une palette de prestataires plus modestes (PME, développeurs indépendants, coopératives) qui proposent des solutions à façon autour de la collecte, l'analyse et la visualisation de données OpenStreetMap.

Passons désormais à la pratique : nous allons manipuler concrètement les données OpenStreetMap, et notamment créer notre premier objet !

Premier contact : éditer les données

Pour commencer, rendez-vous sur le site https://openstreetmap.org/ et créez-vous un compte (bouton « S'inscrire » en haut à droite). Vous devez y renseigner un nom d'utilisateur et une adresse de courriel. Vous recevrez quelques instants plus tard un courriel contenant un lien de validation. Le temps que le message arrive, vous pouvez déjà consulter la carte (qui ne nécessite pas d'inscription). Là où l'on comprend l'intérêt d'une base de données géographique plutôt que d'une simple carte, c'est en ouvrant le volet « Couches » (bouton à droite). Vous pouvez choisir de basculer entre quatre rendus possibles : le standard, la carte cyclable, le fond transports et le rendu humanitaire. Vous remarquerez que selon la couche choisie, les données mises en avant diffèrent. La vocation d'OpenStreetMap est donc bien de proposer des données et non pas des fonds de carte. C'est pour cela qu'il en existe une pléthore mise à disposition par divers fournisseurs.

Maintenant que vous avez reçu le courriel de validation et activé votre compte, vous allez pouvoir modifier les données en elles-mêmes. Attention : vos modifications une fois enregistrées sont mises à disposition de tous, ainsi nous comptons sur votre bienveillance pour alimenter la base avec le plus grand soin ! Cliquez sur le bouton « Modifier » en haut de page. Vous accéderez à l'éditeur web d'OpenStreetMap nommé « iD ». Il s'agit d'un outil développé en JavaScript et notamment basé sur la bibliothèque d3.js pour l'affichage. Je vous invite à suivre le tutoriel interactif affiché à la première utilisation pour bien comprendre son fonctionnement.

Je vous propose de commencer par une modification assez simple : ajouter des détails sur la rue devant chez vous. Rendez-vous dans l'outil sur la zone en question. À noter que dans l'outil, il faut distinguer les données issues d'OpenStreet Map (tout ce qui est cliquable et éditable) des images aériennes en fond (affichées pour faciliter l'édition, mais qui ne font pas partie du projet). Ces images sont mises à disposition par des partenaires et doivent n'être utilisées que pour éditer OpenStreet Map. On constate une richesse des objets vectoriels disponibles : selon la zone, vous aurez les routes, bâtiments, commerces et services renseignés. Si vous cliquez sur un des objets, vous aurez dans le volet de gauche accès aux informations le concernant. Celles-ci sont mises en forme dans l'éditeur pour faciliter la compréhension. En pratique, ce qui est stocké en base correspond aux attributs listés dans « Tous les tags » (dans le bas du volet de gauche). Si vous cliquez sur le bouton « i » à côté de l'un de ces attributs, vous aurez un court descriptif de sa signification. Maintenant, sélectionnez le segment de route de votre choix. Dans le volet de gauche, vous verrez les informations le concernant. Ajoutez par exemple la vitesse maximale autorisée, le nombre de voies ou le revêtement au sol. Une fois fait, cliquez sur le bouton « coche » (en haut du volet de modification). À ce moment, vos modifications sont prises en compte dans votre navigateur, mais la base de données n'est pas encore modifiée. Pour ce faire, il faut cliquer sur « Sauvegarder ». Vous allez renseigner dans le volet de gauche les informations concernant votre groupe de modifications, c'est-à-dire les changements que vous avez réalisés sur la base (à la manière d'un commit sur un système de gestion de versions comme Git). Précisez en particulier un commentaire, comme « Ajout de détails Rue du Marché », qui soit à la fois synthétique et en même temps explicite sur les changements apportés. À noter qu'il faut de préférence écrire ce descriptif dans la langue parlée dans la zone modifiée, ou à défaut en anglais. Puis cliquez sur « Envoyer » pour que la modification soit effective. Voilà, vous avez édité OpenStreetMap, bien joué !

Nous allons désormais passer au niveau supérieur : vous allez ajouter un nouveau point, par exemple un commerce. Cliquez sur le bouton « Point » afin de créer une nouvelle géométrie ponctuelle. À l'aide du curseur, placez le point sur le lieu souhaité. Le volet de gauche vous propose par défaut une typologie non exhaustive pour renseigner ce nouvel endroit. Si le type d'élément désiré n'apparaît pas dans la liste, vous pouvez chercher à l'aide de mots clés dans la barre de recherche. Imaginons que vous souhaitiez indiquer un café, cliquez sur l'entrée correspondante dans la liste. Le lieu est maintenant noté comme tel, et le volet de gauche vous propose d'en renseigner davantage sur l'établissement, de manière identique à la route précédente. On indiquera par exemple son nom, ses horaires d'ouverture, s'il dispose d'un accès Internet, son accessibilité en fauteuil roulant… Une fois fait, vous pouvez valider avec le bouton « Coche » en haut du volet. Et comme précédemment, sauvegardez puis envoyez votre groupe de modification. C'est donc votre deuxième changement sur la base, bientôt la dizaine !

En résumé

Cet article introductif vous a permis de mieux cerner les enjeux autour des données géographiques. Vous avez vu qu'avoir la main sur ces informations permet d'envisager de nombreux cas d'usages. Plusieurs fournisseurs existent, et OpenStreetMap se démarque par son ouverture communautaire et sa richesse sur la description attributaire. Ce projet, similaire à Wikipédia, dispose d'une architecture technique complète, afin d'assurer sa prospérité et sa montée en charge. Vous avez vu comment se composent les données, et êtes en mesure de les modifier. Ces éléments permettent de mieux comprendre cet écosystème riche, d'en avoir une meilleure vue d'ensemble. Ainsi, nous pourrons mieux aborder les mises en pratique techniques lors des prochains articles de cette série. Et notamment comment créer sa propre carte interactive en utilisant un fond OpenStreetMap personnalisé avec sa palette de couleurs. •

Pour aller plus loin

Site d'OpenStreetMap :
https://openstreetmap.org
Documentation :
https://wiki.openstreetmap.org
Tutoriels de découverte : https://learnosm.org
Tutoriels techniques : https://switch2osm.org

Adrien Clerbois
Lead Design @ Ingenico
Microsoft MVP Developer technologies
Twitter : @AClerbois

Du temps réel dans vos applications avec ASP.NET Core SignalR

niveau 100

Avez-vous toujours eu envie de réaliser un chat ? Avec un client qui émet un message et des clients qui reçoivent des communications en temps réel ? Le pari n'est pas risqué, si l'on s'équipe du bon outil : ASP.NET Core SignalR.

Qu'est-ce que SignalR ?

ASP.NET Core SignalR est une bibliothèque open-source qui, une fois ajoutée à son projet, simplifie l'ajout de fonctionnalités temps réel à vos applications. La fonctionnalité Web en temps réel permet au code côté serveur de pousser le contenu vers les clients instantanément et inversement.

Voici quelques caractéristiques de SignalR pour ASP.NET Core :

- Gestion automatique des connexions.
- Envoi des messages à tous les clients connectés simultanément.
- Envoi des messages à des clients ou groupes de clients spécifiques.
- Mise à l'échelle pour faire face à l'augmentation du trafic.

Code source de SignalR: https://github.com/aspnet/signalr

La communication

SignalR fournit une API pour la création d'appels de procédures à distance de serveur à client (RPC). Voici les différentes façons que SignalR utilise pour pouvoir communiquer entre le client et le serveur :

- WebSockets : cette technique vise à développer un canal de communication full-duplex sur un socket TCP entre le client et le serveur.
- Server-Sent Events : il s'agit d'une technologie où un navigateur reçoit des mises à jour automatiques d'un serveur via une connexion HTTP.
- Long Polling : le client émet une connexion avec le serveur et le serveur relâche la connexion une fois qu'il a une réponse à lui donner.

SignalR détermine le protocole qui convient le plus à chaque client par élimination. Il commence par le WebSocket, ensuite le Server-Sent Events et enfin de Long Polling.

Hubs

SignalR utilise comme point d'entrée les Hubs afin de relier les clients et les serveurs.

Un hub est un pipeline de haut niveau qui permet à un client et à un serveur d'appeler des méthodes l'une sur l'autre. SignalR gère automatiquement la répartition au-delà des limites de la machine, ce qui permet aux clients d'appeler des méthodes sur le serveur et vice versa. Vous pouvez passer des paramètres fortement typés à des méthodes, ce qui permet de lier le modèle.

SignalR fournit deux protocoles de hub intégrés : un protocole texte basé sur JSON et un protocole binaire basé sur MessagePack.

MessagePack crée généralement des messages plus petits que JSON. Les navigateurs plus anciens doivent prendre en charge XHR niveau 2 pour assurer la prise en charge du protocole MessagePack.

Implémentation

Afin de pouvoir suivre cet exemple, vous devez avoir sur votre machine :

- Visual Studio Code (https://code.visualstudio.com/download) ;
- .NET Core SDK 2.1 et plus (https://www.microsoft.com/net/download/archives) ;
- C# pour Visual Studio Code (http://bit.ly/CSharpForVSCode) ;
- Node.JS et npm (https://www.npmjs.com/get-npm).

Créons le projet. Pour cela, ouvrez un dossier que vous allez utiliser pour le projet et dans un Command Line, exécutez la commande suivante :

```
dotnet new webapp -o aclerbois.programmez.chat
```

```
c:\programmez
λ dotnet new webapp -o aclerbois.programmez.chat
Getting ready...
The template "ASP.NET Core Web App" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/aspnetcore-template-3pn-210
 for details.

Processing post-creation actions...
Running 'dotnet restore' on aclerbois.programmez.chat\aclerbois.programmez.chat.csproj...
  Restoring packages for c:\programmez\aclerbois.programmez.chat\aclerbois.programmez.chat.csproj...
  Generating MSBuild file c:\programmez\aclerbois.programmez.chat\obj\aclerbois.programmez.chat.csproj.nuget.g.props.

  Generating MSBuild file c:\programmez\aclerbois.programmez.chat\obj\aclerbois.programmez.chat.csproj.nuget.g.target
s.
  Restore completed in 9.63 sec for c:\programmez\aclerbois.programmez.chat\aclerbois.programmez.chat.csproj.

Restore succeeded.
```

La bibliothèque de serveur SignalR est incluse dans le méta paquet Microsoft.AspNetCore.App. Pour la partie client, vous devez télécharger la bibliothèque JavaScript SignalR à partir de npm, le gestionnaire de paquets Node.js. Rendez-vous dans le dossier du projet :

```
cd aclerbois.programmez.chat\
```

Lançons l'initialisation de npm pour créer le fichier de configuration package.json :

```
npm init -y
```

```
Wrote to c:\programmez\aclerbois.programmez.chat\package.json:

{
  "name": "aclerbois.programmez.chat",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Installons la bibliothèque SignalR :

```
npm install @aspnet/signalr
```

```
+ @aspnet/signalr@1.0.3
added 1 package in 5.889s
```

La commande npm install lance le téléchargement de la bibliothèque client en JavaScript dans le répertoire contenant les modules node.js. Il est important de reprendre le contenu installé pour l'insérer dans le fichier wwwroot/lib.

```
mkdir "c:\programmez\aclerbois.programmez.chat\wwwroot\lib\signalr"
cp node_modules\@aspnet\signalr\dist\browser\signalr.js wwwroot\lib\signalr\
```

Notre premier hub SignalR

Un hub est une classe qui sert de pipeline de haut niveau. Il gère la communication client-serveur.
Nous allons créer un dossier Hubs pour centraliser tous les hubs disponibles dans notre application :

```
mkdir Hubs
```

Dans ce nouveau répertoire fraîchement créé, ajoutons le fichier ChatHub.cs avec le code suivant :

```
using Microsoft.AspNetCore.SignalR;
using System.Threading.Tasks;

namespace SignalRChat.Hubs
{
  public class ChatHub : Hub
  {
    public async Task SendMessage(string user, string message)
    {
      await Clients.All.SendAsync("ReceiveMessage", user, message);
    }
  }
}
```

La classe ChatHub hérite de la classe SignalR Hub qui gère les connexions, les groupes et le messaging.
La méthode SendMessage peut être appelée par n'importe quel client connecté. Il émet le message à tous les clients. SignalR utilise l'asynchronisme afin de maximiser la mise à l'échelle.

Configurer le projet pour utiliser SignalR

Dans le fichier Startup.cs, ajoutons le middleware SignalR dans le pipeline applicatif ainsi que dans le système d'injection de dépendances :

```
...
using aclerbois.programmez.chat.Hubs;

namespace aclerbois.programmez.chat
{
  public class Startup
  {
    ...

    public void ConfigureServices(IServiceCollection services)
    {
      ...
      services.AddSignalR();
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
      ...
      app.UseSignalR(routes =>
      {
        routes.MapHub<ChatHub>("/chatHub");
      });
      app.UseMvc();
    }
  }
}
```

Le client

Remplaçons le contenu du fichier Pages/Index.cshtml avec :
Voir code complet sur www.programmez.com
Nous avons, à l'aide de ce code, créé des zones de texte pour le nom, le texte du message et un bouton, qui permet de soumettre notre message, ainsi qu'une liste qui nous permettra d'afficher les messages en provenance du hub SignalR. Pour finir, ajoutons le script SignalR et le script de logique applicative pour la gestion du Chat, chat.js. Ce dernier n'a pas encore été créé ; créons-le dans le répertoire wwwroot/js et insérons le code suivant :
Voir code complet sur www.programmez.com
Ce script permet d'établir une connexion avec le hub, le démarre et lie le bouton pour envoyer le message au hub. Lions également l'évènement ReceiveMessage, appelé à chaque nouveau message reçu, et affichons-le dans la liste.

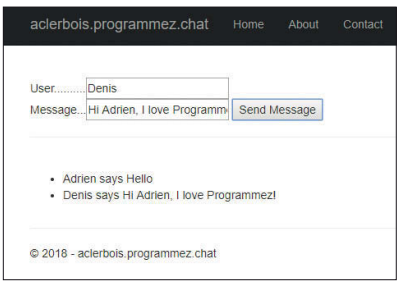
Lancer l'application

Vous n'avez plus qu'à exécuter l'application et votre chat apparaîtra :

```
dotnet run
```

```
c:\programmez\aclerbois.programmez.chat  (aclerbois.programmez.chat@1.0.0)
λ dotnet run
Using launch settings from c:\programmez\aclerbois.programmez.chat\Properties\launchSettings.json...
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Users\Adrien\AppData\Local\ASP.NET\DataProtection-Keys' as key repository
and Windows DPAPI to encrypt keys at rest.
Hosting environment: Development
Content root path: c:\programmez\aclerbois.programmez.chat
Now listening on: https://localhost:5001
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

Et voici le résultat dans notre navigateur : ci-dessous
Vous pouvez trouver les sources sur mon GitHub : https://github.com/aclerbois/aclerbois.programmez.signalr



Donnez de la vie à vos applications

Grâce à SignalR, vous allez pouvoir donner de la vie à vos applications, et ce facilement grâce à l'intégration native de SignalR dans l'ASP.NET Core 2.1.
J'ai utilisé cette bibliothèque pour créer une application de tirage au sort pour l'édition 2017 du DevDay (http://www.devday.be).
La mise en application du temps réel pour mon application convient parfaitement pour le projet. Cependant je n'aurais jamais imaginé que la connexion internet était limitée en nombre de connexions simultanées autorisées. Résultat : après 50 connexions WebSocket, le routeur nous a bloqué et j'ai dégradé le mode de connexion pour le deuxième essai en mode Polling. Finalement, la tombola s'est très bien terminée.

> Je vous invite à nous rejoindre pour le DevDay 2018 à Louvain la Neuve, le 27 novembre 2018, pour que vous puissiez, vous aussi, me parler de vos expériences sur le monde .NET.

Jordan NOURRY
Développeur
@La combe du lion vert

Fouad JADOUANI
Développeur
@Société Générale

Refactoring de Legacy Code avec Programmation Fonctionnelle en pur JavaScript

Partie 1

La programmation fonctionnelle est devenue un sujet très tendance ces dernières années. Complètement méconnue des développeurs débutants, pour la plupart concentrés sur la programmation orientée objet, elle permet de rendre le code plus concis, plus simple et plus expressif.
Comment appréhender ce style de programmation ? Quels sont ses points forts ? Et surtout comment « refactorer » du code legacy en y ajoutant de la programmation fonctionnelle ? Pour étayer nos propos nous allons nous appuyer sur le langage JavaScript, pour lequel la communauté a énormément travaillé ces 3 dernières années à l'intégration de la programmation fonctionnelle dans notre environnement professionnel !

niveau 200

La programmation fonctionnelle est un paradigme de programmation de type déclaratif qui considère le calcul en tant qu'évaluation de fonctions mathématiques. Pour vulgariser cette définition, la programmation fonctionnelle est une méthode de développement par composition en fonction pure, ce qui vous permet d'éviter la mutation des données (mutable data), les états partagés (shared state) et les effets de bord (side-effects).

UN PEU D'HISTOIRE

La programmation impérative est un des styles de programmation les plus connus, issue de la machine de Turing qui est un modèle abstrait du fonctionnement des appareils mécaniques de calcul. Ce modèle est basé sur le changement d'état et donc la création d'effet de bord. Pourtant quelques années avant la présentation des travaux d'Alan Turing sur la machine de Turing **en 1936, le mathématicien Alonzo Church présentait ses travaux qui fondent les concepts de fonction** et d'application.

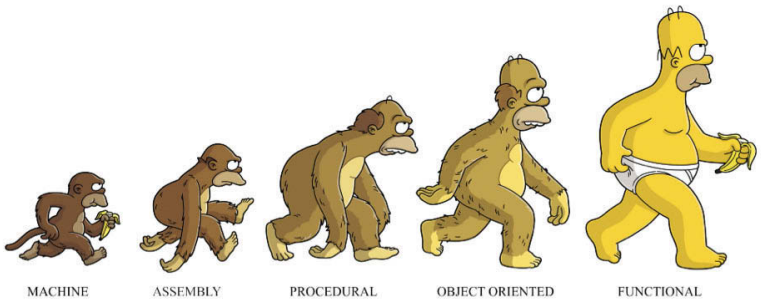
La programmation fonctionnelle est donc issue du modèle de Church et est **basée sur les Lambda calculs**. Les Lambdas calculs ont été développés en 1930 et la pensée se veut plus fonctionnelle et **hérite du côté pur des mathématiques**, d'où provient l'immutabilité, car une valeur en mathématique ne bouge pas.

Pas étonnant que le premier langage permettant la programmation fonctionnelle ait été utilisé principalement pour le calcul scientifique ; il s'agit de ***Fortran* (FORmula TRANslator)** en 1954, suivi de sa deuxième version en 1958.

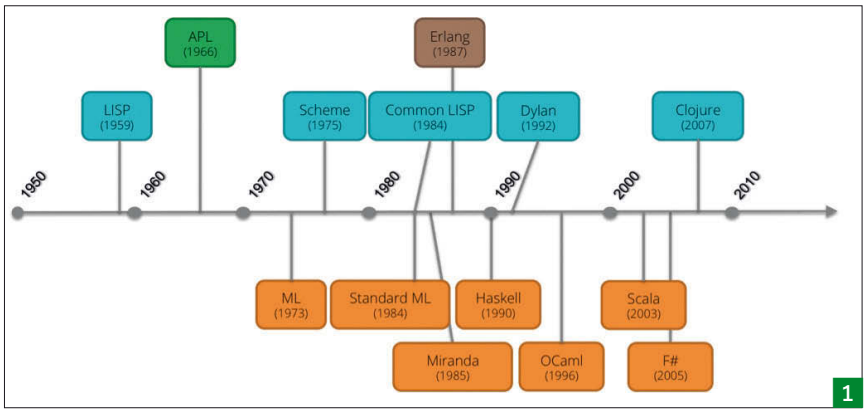
En 1959 on retrouve le langage de John McCarthy, ***Lisp*** qui permet de faire de la programmation fonctionnelle. Cependant le paradigme impératif est aussi possible.

Dérivé du langage Lisp, ***Scheme*** créé en 1975 est un langage de programmation qui se veut être un Lisp plus épuré, en conservant uniquement les aspects essentiels : la flexibilité et la puissance expressive. En gros, seul le paradigme fonctionnel est possible avec ce langage.

Dans le genre pur fonctionnel, on retrouve le fameux langage ***Haskell***, qui est un langage **influencé par Lisp et Scheme**, qui a été créé en 1990. Haskell est fondé sur le Lambda-calcul et la logique



SOURCE IMAGE : http://www.olympp.fr/programmation-fonctionnelle-avec-scala/



1

combinatoire. Son nom vient du mathématicien et logicien Haskell Brooks Curry; nous lui devons aussi le nom du concept de "Curryfication" de fonction. Une nouvelle version plus récente de Haskell est disponible depuis 2010. Il s'agit d'une version minimale et portable du langage, conçue à des fins pédagogiques et pratiques.

Entre le trio Lisp, Scheme et Haskell, on retrouve d'autres langages comme **OCaml, anciennement connu sous le nom d'Objective Caml,** dont la première version CAML a été créée en 1987 et est multi-paradigme : impérative, fonctionnelle et orientée objet. OCaml possède la plupart des caractéristiques communes aux langages fonctionnels, en particulier des fonctions d'ordre supérieur et fermetures (closures), ainsi qu'un bon support de la récursion ter-

Abonnez-vous à Programmez! Abonnez-vous à Programmez! Abonne

[Programmez!] Le magazine des développeurs

Nos classiques

1 an
11 numéros
49€*

2 ans
22 numéros
79€*

Etudiant
1 an - 11 numéros
39€*

* Tarifs France métropolitaine

Abonnement numérique

PDF ..... **35€**
1 an - 11 numéros
Souscription uniquement sur www.programmez.com
Option : accès aux archives 10€

Offres 2018

1 an ..... **59€**
11 numéros
+ 1 vidéo ENI au choix :

- Symfony 3*
  Développer des applications web robustes
  (valeur : 29,99 €)
- Raspberry Pi*
  Apprenez à réaliser et piloter une lumière d'ambiance
  (valeur : 29,99 €)

2 ans ..... **89€**
22 numéros
+ 1 vidéo ENI au choix :

- Symfony 3*
  Développer des applications web robustes
  (valeur : 29,99 €)
- Raspberry Pi*
  Apprenez à réaliser et piloter une lumière d'ambiance
  (valeur : 29,99 €)

* Offre limitée à la France métropolitaine

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.

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

☐ **Abonnement 1 an :** 59 €
11 numéros + 1 vidéo ENI au choix :
☐ **Abonnement 2 ans :** 89 €
22 numéros + 1 vidéo ENI au choix :

☐ Vidéo : Symfony 3
☐ Vidéo : Raspberry Pi

PROG 222
Offre limitée, valable jusqu'au 2 novembre 2018

☐ Mme ☐ M. 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**

Offres 20e anniversaire !*

1 an (11 numéros)

79,99 €

(au lieu de 147,99 €)

2 ans (22 numéros)

99,99 €

(au lieu de 177,99 €)

+

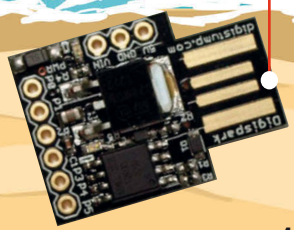
1 clé USB contenant tous les numéros depuis le n°100

+

4 numéros vintages (selon les stocks)

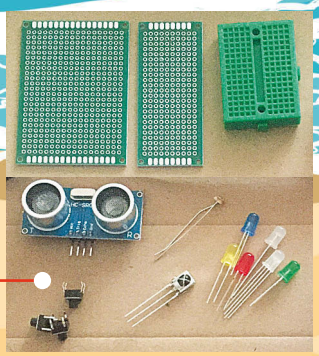
+

1 carte maker Digispark ATtiny84 compatible Arduino



+

1 lot de composants / capteurs (selon arrivage du jour)



+

1 an de Pharaon Magazine soit 4 numéros : pour découvrir l'Egypte des Pharaons !

** Offre réservée à la France métropolitaine*

Sur notre site web uniquement : **https://www.programmez.com/catalog/20eme-anniversaire**

minale.Voici une petite timeline de récapitulation de création des langages de programmation fonctionnelle. 1

SOURCE:
https://vampwillow.wordpress.com/2016/04/17/adopting-functional-programming-languages-part-3/
https://fr.wikipedia.org/wiki/Programmation_impérative
https://fr.wikipedia.org/wiki/Lambda-calcul
https://fr.wikipedia.org/wiki/Haskell

JAVASCRIPT vs LES AUTRES LANGAGES DE PROGRAMMATION

Pourquoi avoir choisi Javascript pour vous parler de programmation fonctionnelle ? Car nous le savons tous, nous sommes très loin d'un langage pur. Cependant ce n'est pas un hasard, Javascript a de nombreux avantages, et comme **Douglas Crockford**, auteur du livre « Javascript : les bons éléments », le disait déjà en 2012, Javascript est un langage qui va exploser dans les années à venir ! (https://www.youtube.com/watch?v=dkZFtimgAcM)
Martin Fowler lui-même, dans un récent tweet, a annoncé la sortie de la seconde édition de son livre "Refactoring : Improving the Design of Existing Code" avec des exemples de code en Javascript, et non plus en Java comme dans la première édition. Comme il le dit dans son article de blog sur le choix de ce langage :

> *"ECMAScript 2015 (ES6) a introduit un modèle de classe plutôt bon, qui rend beaucoup plus facile l'expression de nombreux « refactorings » orientés objet [...] la raison de le choisir à Java est qu'il n'est pas entièrement centré sur les classes. Il existe des fonctions de niveau supérieur et l'utilisation de fonctions de première classe est courante. Cela rend beaucoup plus facil d'afficher le « refactoring » hors du contexte des classes."*

(https://martinfowler.com/articles/refactoring-2nd-ed.html)

Effectivement, pour comprendre toutes les subtilités de la programmation fonctionnelle et ne pas retomber dans les travers de fonctions impures à la première difficulté, **la communauté prône Haskell comme langage d'apprentissage**. Mais comme nous allons le détailler dans la suite de cet article, si vous développez en ES6, vous avez déjà un pied dans la programmation fonctionnelle sans le savoir.

LES CONCEPTS LES PLUS PUISSANTS

La courbe d'apprentissage est courte une fois les concepts de base assimilés. Nous allons faire le tour de tous les concepts les plus puissants un à un. Cela vous donnera une vue d'ensemble du style de la programmation fonctionnelle.
Par la suite nous utiliserons tous ces concepts dans un exemple concret, avec un refactoring d'un code Javascript écrit dans un style "rétro", en programmation fonctionnelle.
Allez, on commence par les bases !

LES FONCTIONS

Dans le paradigme de la programmation fonctionnelle l'unité de traitement est la **fonction**. Nous allons donc commencer par énoncer la différence majeure qui existe entre la plupart des fonctions que nous trouvons en programmation procédurale et celles que nous trouvons en programmation fonctionnelle.

IMPURE VS PURE FONCTION

Comme évoqué précédemment, la programmation fonctionnelle vient des mathématiques et les fonctions arithmétiques sont l'archétype des fonctions pures. Une fonction pure est une fonction dont le résultat ne dépend que des arguments et non d'un état externe, et qui n'a pas d'effets de bord, c'est-à-dire d'effets secondaires tels que des changements d'états.
Voici un exemple de fonction impure :

```
let value = 5;

const add = function() {
  value += 1;
  return value;
};

add(); // <-- 6
add(); // <-- 7
add(); // <-- 8
```

L'appel répété de la fonction *add* sans argument, engendre une mutation de la valeur de la variable *value* en interne, donc à chaque appel elle est incrémentée de 1, alors qu'elle n'est pas passée en argument de la fonction.
Voici le même algorithme en fonction pure :

```
const add = function(value) {
  return value + 1;
};

add(5); // <-- 6
add(5); // <-- 6
add(5); // <-- 6
```

Dans cette version, la variable *value* est explicitement passée en paramètre à la fonction et la fonction ne fait pas de mutation sur cette variable. A la place, elle renvoie une nouvelle référence qui est le résultat du calcul. Donc l'appel répété de **cette fonction donnera toujours le même résultat tant qu'on lui passe toujours le même argument en entrée.**
Chose qui n'était pas vraie avec **la fonction impure, car elle ne respecte pas la transparence référentielle, ni l'immutabilité et donc génère des effets de bords.**
On décortique toutes ces notions tout de suite !

LA TRANSPARENCE RÉFÉRENTIELLE

Comme dit auparavant, la transparence référentielle est un principe selon lequel le résultat retourné par une fonction est toujours le même chaque fois qu'elle est appelée avec les mêmes paramètres.

```
let counter = 0;
function incrementCounter(c) {
   counter = c + 1;
```

```
    notifySomeone(c);
    return counter;
}
incrementCounter(counter) === incrementCounter(counter);
// 1 === 2
// ← false
```

Dans l'exemple on voit qu'en dehors du fait que la fonction *incrementCounter* change une variable qui ne fait pas partie de son scope, elle appelle aussi une méthode *notifySomeone* qui elle-même peut probablement modifier le contexte extérieur de la fonction. On dit que la fonction comporte des **effets de bord**.

```
function increment(c) {
    return c + 1;
}

increment(2) === increment(2);
// 3 === 3
// ← true
```

Dans cet exemple, la fonction *incrément* est dite **fonction pure** car elle est indépendante du contexte extérieur.
L'avantage de ce concept est qu'il vous permettra de :

- Décomposer et découper un problème complexe en plusieurs petites parties moins compliquées, ce qui permet d'avoir un code simple à tester et facile à comprendre.
- Optimisation du code via *memoization(1)* qui permet de réduire le temps d'exécution d'un appel à une fonction avec les mêmes entrées.
- Parallélisation d'exécution (*thread safe*) car la fonction ne dépend pas du contexte extérieur pour faire son traitement.

L'IMMUTABILITÉ

L'immutabilité est un des prérequis nécessaires pour faire de la programmation fonctionnelle où les valeurs des variables ne changent pas une fois attribuées. En effet, la mutation d'un élément peut souvent entraîner des changements de comportement des éléments sous-jacents.
L'immutabilité a le mérite de **simplifier la gestion d'état de l'application** en réduisant le nombre de bugs qui peuvent être introduits avec une mutation.
En JavaScript, il est important de comprendre que « **const** » ne veut pas dire constante ou immutable. Mais cela signifie que l'on crée une référence en lecture seule vers une valeur qu'on ne peut pas modifier. Ainsi, si votre variable est déclarée en tant que primitive, alors elle sera immutable.

```
const author = 'Jordan';
author = 'Fouad';
// Exception TypeError: invalid assignment to const `h'
```

En revanche, si votre variable est un objet, alors elle sera mutable car elle est seulement une référence vers l'objet et les propriétés de cet objet ne font pas partie de la référence.

```
const article = {author: 'Jordan'};
article.author = 'Fouad';
console.log('Author:', article.author);
// ← Author: Fouad
```

En Javascript, pour déclarer une variable immutable on peut utiliser la fonction **freeze**.

```
const article = Object.freeze({author: 'Jordan'});
article.author = 'Fouad'
// Exception TypeError in strict mode;
// silencieux en sloppy mode
```

Cependant cette méthode ne rend immuable que le premier niveau de l'objet, si celui-ci est complexe avec plus d'un niveau d'imbrication alors il sera mutable.

```
const article = {author: {name: 'Jordan', mail: 'mail@sgcib.com'}};
article.author.name = 'Fouad';
article.author.mail = 'mail@sgcib.com';
console.log('Author:', article.author.name, 'Mail:',
article.author.mail);
```

Pour résoudre ce problème, le plus simple est d'utiliser un outil tel que : *deep-freeze(2)* ou *freezer(3)*, ou une librairie telle que ImmutableJS*(4)* ou encore mori*(5)* qui sont proposées par la communauté pour la manipulation des objets immutables.

LES EFFETS DE BORDS

Un effet de bord (effet secondaire) consiste à modifier l'état interne d'un programme, suite à une opération telle que :

- Modifier une variable statique ou globale ;
- Modifier un de ses arguments ;
- Faire un appel (http, base de données…) ;
- Lancer une exception ;
- Afficher un log.

```
let author = {name: 'Fouad'};
function setAuthor(_author) {
 author = _author;
 authorRepository.update(author);
 return author;
}
```

La fonction *setAuthor* modifie une variable *author* et appelle une autre *fonction authorRepository.update(author)* qui fait partie d'un contexte plus large que le sien, ce qui rend la fonction indéterministe. Le paradigme fonctionnel préconise d'éviter les effets de bord. En effet, cela évite d'avoir à faire un raisonnement local sur la fonction, réduit le couplage, et facilite la composition et la testabilité. Voyons à présent plus en détail dans la partie suivante les avantages des fonctions pures.

(1) https://fr.wikipedia.org/wiki/M%C3%A9mo%C3%AFsation

(2) https://github.com/substack/deep-freeze/
(3) https://github.com/arqex/freezer
(4) https://facebook.github.io/immutable-js/
(5) http://swannodette.github.io/mori/

LES AVANTAGES DES FONCTIONS PURES

Le fait de respecter les trois concepts abordés précédemment, qui constituent la notion de pure fonction, ont a eux seuls de grande vertus.

LES PURES FONCTIONS SE PARALLÉLISENT BIEN

Quand nous avons un nombre assez volumineux de données à traiter et que ces données peuvent être traitées séparément, nous avons tendance à opter pour un traitement parallèle et ainsi à utiliser la pleine puissance de nos processeurs actuels.
Le problème est que les fonctions pures et impures ne sont pas égales face à la parallélisation. **Les fonctions pures se parallélisent bien mieux et, plus généralement, sont immunisées contre les problèmes qui rendent la concurrence difficile**.

TESTABILITÉ

Les fonctions pures ont pour avantage d'être prédictibles, grâce à la transparence référentielle. Ceci permet de les tester plus facilement et surtout de mettre leur résultat en cache afin ne pas avoir à refaire le calcul pour des valeurs déjà traitées : ce qui est possible grâce à la **memoïzation**.

REFACTORING

Comme évoqué précédemment, ce n'est pas un hasard si Martin Fowler prépare la 2e édition de son livre parlant de « Refactoring » en Javascript, et il y a fort à parier qu'il y aura de la programmation fonctionnelle. L'absence d'effets de bords, en s'assurant qu'une fonction retournera toujours la même valeur de sortie pour un même ensemble de données d'entrée, est une règle qui permet d'arriver facilement à la programmation fonctionnelle. L'une des manières de basculer vers la programmation fonctionnelle consistant à refactoriser le code existant afin d'éliminer les effets secondaires indésirables.

SOURCE :
https://hackernoon.com/functional-programming-in-c-purity-and-concurrency-cd7835a1986a
https://medium.com/@hkairi/programmation-fonctionnelle-avec-javascript-df4dafdbecda

CITOYEN DE PREMIÈRE CLASSE

En Javascript, les fonctions sont des **objets de première classe**(6) (First-class citizen), cela veut dire qu'une fonction peut :
- Être affectée à des variables ou des structures de données ;
- Être passée comme paramètre à une fonction ;
- Être retournée par une fonction.

Ce qui explique certainement pourquoi Javascript est devenu incontournable quand on parle de programmation fonctionnelle.

ASSIGNATION DE FONCTIONS

En Javascript nous pouvons assigner une fonction avec plusieurs signatures légèrement différentes :

```
const myFunction = function() {
    // ... le corps de ma fonction
}
```

Ici nous utilisons une fonction anonyme pour l'assignation, mais nous pourrions tout à fait lui donner un nom :

```
const myFunction = function functionName() {
    // ... le corps de ma fonction
}
```

On peut également lui passer des paramètres supplémentaires :

```
const myFunction = function functionName(param) {
    // ... le corps de ma fonction;
}
```

Les deux autres concepts définissant un citoyen de première classe introduisent directement l'un des concepts clés le plus puissant quand on parle de programmation fonctionnelle ; il s'agit des fonctions d'ordre supérieur.

LES FONCTIONS D'ORDRE SUPÉRIEUR

Une **fonction d'ordre supérieur**(7) est une fonction qui a au moins une des propriétés suivantes :
- Elle prend une ou plusieurs fonctions en entrée ;
- Elle renvoie une function.

```
const addArticlesToAuthor1 = author => articles => ({...author, articles});

function addArticlesToAuthor2(author) {
  return function(articles) {
    return ({...author, articles});
  }
}
// addArticlesToAuthor1 et addArticlesToAuthor2 sont équivalentes, mais écrites
 de deux façon différentes.
// Nous préférons la première syntaxe qui consiste à utiliser les lambdas expressions.
```

Les lambdas(8) sont des fonctions anonymes utilisées de manière ponctuelle et n'effectuent généralement qu'une seule opération.
Dans cet exemple notre fonction *addArticlesToAuthor* reçoit une valeur *author* puis retourne une fonction qui attend *articles* qui par la suite retourne un objet de l'auteur avec ces articles.

```
const jordanAuthor = {name: 'Jordan Nourry'};
const jordanArticles = ['Ton Code fait partie du Patrimoine Culturel'];

const addArticlesToJordan = addArticlesToAuthor(jordanAuthor);
addArticlesToJordan(jordanArticles);
// <- {name: 'Jordan Nourry', articles: ['Ton Code fait partie du Patrimoine Culturel']}
```

Comme vous pouvez le constater dans cet exemple, l'intention d'ajouter des articles à un auteur est explicite, ce qui rend le code facile à comprendre car il cache les détails de l'opération.
Donc l'intérêt des fonctions d'ordre supérieur est l'expressivité du code, en mettant l'accent sur la tâche à accomplir plutôt que sur la façon de l'accomplir.
https://fr.wikipedia.org/wiki/Objet_de_première_classe

La suite de l'article le mois prochain

(6) http://ryanchristiani.com/functions-as-first-class-citizens-in-javascript/

(7) https://fr.wikipedia.org/wiki/Fonction_d'ordre_supérieur
(8) https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Fonctions/Fonctions_fléchées

Jean-Baptiste Cazaux.
Développeur & formateur react.

React : l'alternative à Angular que vous devez utiliser !

Partie 1

ReactJs est une librairie écrite par Facebook et rendue publique en 2013. Elle permet de créer des applications dans les navigateurs web. Elle est en concurrence directe avec Angular de Google et Vue.js.

Il existe depuis 3 ans react-native, qui permet créer des applications iOS et Android. React-native reprend la façon d'écrire des applications de ReactJs, mais avec des composants graphiques adaptés au mobile. Depuis 1 an, Facebook propose aussi react VR, pour développer des applications de VR. Reactjs, react-native et reactVR ont comme socle commun react-core.

React est utilisé par les sociétés majeures, comme pour des applications plus modestes. Parmi les plus connues on compte bien sûr Facebook et Instagram, mais aussi Twitter, Airbnb, Netflix, Dropbox… 1 2

Avant de se lancer

React propose un cadre de développement, mais cela reste une librairie qui limite sciemment le périmètre qu'elle se veut gérer.

Il faut donc choisir les librairies qui vont être complémentaires à React. Et même pour des applications simples il y a beaucoup de décisions à prendre !

On va par exemple avoir besoin de faire des appels réseaux, utiliser des composants graphiques, gérer les URLs (routing) et l'historique de navigation, faire des tests, …

Quand on choisit de faire du react, on a donc juste commencé le processus de réflexion !

Avec Angular, la pile technique vient plutôt d'un bloc sans que l'on ait vraiment à se poser de questions, même s'il est toujours possible de changer certaines briques grâce à l'injection de dépendance.

Et lorsque l'on a choisi les librairies, ce n'est pas fini ! Comme avec Angular et Vue, on va pouvoir écrire en Javascript "classique" (ES5), en ES6, ES7.., ou encore du Javascript typé comme TypeScript ou Flow. Bien sûr, il y a une transpilation du code avec Babel afin que l'application puisse être exécutée dans des navigateurs plus ou moins récents (on peut spécifier à Babel un nombre réduit de versions récentes, afin de réduire la taille du code généré).

Un des autres points très différenciants est la façon d'écrire le code HTML qui sera généré. Si en Vue et en Angular, on écrit le template html en dehors du composant, en react, le html est dans le composant. Cette syntaxe particulière s'appelle JSX. 3

Il y a deux façons de créer un composant React. La première est d'étendre la classe React.Component et d'implémenter la méthode *render()*. Un *render()* classique retourne un ou plusieurs éléments HTML ou composants React, décrits en JSX. La seconde est d'écrire une méthode qui renvoie du JSX.

Composants

Comme avec Angular et Vue, React est orienté composants. C'est à dire que l'on va découper l'application en une multitude de composants simples, réutilisables et paramétrables.

Un composant est par exemple un menu, qui va lui-même contenir des sous menus qui contiennent chacun des boutons. On se retrouve donc avec une application qui peut être représentée sous forme d'arbre dont les noeuds sont ces composants. Il n'y a qu'une racine, un unique composant qui encapsule tous les autres. 4 5

Bien sûr, il n'y a pas que des composants graphiques. Afin de séparer les responsabilités et donc qu'un composant ne soit pas chargé de gérer la logique d'accès aux données, la logique métier et encore la logique d'affichage, on a des composants qui sont spécialisés.

Comme avec Angular, il peut également y avoir des beans *'service'*, par exemple quand on utilise le pattern *flux/redux*.

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>React !</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react/16.3.2/umd/react.development.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/react-dom/16.3.2/umd/react-dom.development.js"></script>
<link rel="shortcut icon" href="/favicon.ico"></head>
<body>
    <div id="root"></div>
    <script type="text/javascript" src="/bundle.js"></script>
</body>
</html>
```

1

Le fichier html avec les scripts React à charger, bundle.js est le code de l'application HelloWorld converti en ES5.

```
import React from 'react';
import ReactDOM from 'react-dom';

class Hello extends React.Component {
    render() {
        // Les propriétés sont accessibles dans this.props
        return <div>
            Hello {this.props.name}
        </div>;
    }
}


// point d'entrée de l'application: on indique le composant
// racine et dans quel élément HTML on doit le passer
ReactDOM.render(
    <Hello name="world"/>, // on passe une propriété 'name' avec la valeur 'world'
    document.getElementById('root')
);
```

2

L'application HelloWorld, écrite en ES6.

Gestion des données

Dans une application, il y a nécessairement des données à afficher et à manipuler. Ces données sont propres à un composant ou peuvent être partagées entre différents composants.

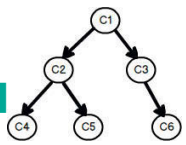
React introduit deux concepts : l'état (*state*) et les propriétés (*props*).

L'état est un objet contenant un ensemble de clés/valeurs, les valeurs étant des chaînes de caractères, des nombres, des tableaux, des objets…

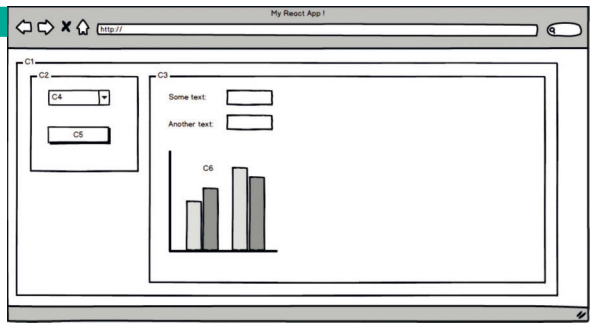
L'état permet de gérer les données internes au composant, tandis que les propriétés sont des attributs passés d'un composant parent à un composant enfant, lors de la déclaration en JSX. 6

Chaque fois que l'état ou les propriétés d'un composant changent, React fait un appel à la méthode *render()*, ce composant et ses enfants sont donc redessinés. Il y a beaucoup d'optimisations pour ne pas redessiner toute la page au moindre changement !

5



4



On n'utilise pas ou peu de variables d'instance d'un composant (un composant est une classe), pour les données que l'on affiche, sinon il faudrait prévenir nous-même React que la variable a changé et qu'il faut donc redessiner le composant.

Sans surprise, on peut se brancher à des moments clés du cycle de vie d'un composant : sa création, sa destruction, quand les propriétés envoyées par le parent changent, … Quand un composant n'a pas d'état, et qu'on ne cherche pas à se servir de ses méthodes du cycle de vie, cela revient à n'écrire qu'une simple fonction *render()*. C'est donc dans ce cas-là que l'on peut écrire un composant sous forme de fonction, qui prend en paramètre les *props* et qui retourne du JSX. 7

Autour de react

React est une base solide, mais il est impossible de se passer des librairies qui gravitent autour, à moins de les réécrire soi-même ! L'idée est que l'on va pouvoir choisir laquelle.

Si l'on veut éviter de recréer la roue, on peut commencer par utiliser une librairie de composants graphiques. Il en existe énormément : *material-ui*, *semantic-ui*,... Le choix dépend évidemment du style de l'application. Pour créer ses propres composants graphiques, il y a une librairie qui facilite la tâche, notamment grâce à un style d'écriture élégant : *styled-components*.

En dehors des librairies à utiliser avec React, il y a également tout un écosystème d'outils.

Les IDE comme Visual Studio ou IntelliJ gèrent parfaitement React et la syntaxe JSX, ce qui permet d'avoir toutes les facilités auxquelles on peut s'attendre dans le développement moderne d'applications.

Comme en Angular, un outil permet de créer un projet minimaliste en React : *create-react-app*. Quelques choix classiques d'outils (webpack, jest, …) et de configurations font en sorte que les premières heures soient passées à comprendre React plutôt que sur les sujets connexes. Create-react-app prend en charge beaucoup de besoins et il est toujours possible d'extraire les fichiers de configuration pour les customiser.

En plus de cela des kits de configuration existent sur des choix différenciants comme reactjs/react-native, es6/Typescript, …

Isomorphisme

Une des fonctionnalités de React de plus en plus utilisée est l'*isomorphisme*. C'est un procédé qui consiste à pouvoir faire le rendu html d'une application React dans le navigateur mais aussi côté serveur.

Générer du html depuis un serveur nodejs (*Server Side Rendering*) permet d'avoir un premier rendu de l'application plus

```jsx
import React from 'react';
import ReactDOM from 'react-dom';

class UserInfos extends React.Component {
    render() {
        return (
            <div className="infos">
                <label>
                    Nom:
                    <input type="text" readOnly={true} value={this.props.user.lastname}/>
                </label>
                <label>
                    Prénom:
                    <input type="text" readOnly={true} value={this.props.user.firstname}/>
                </label>
                <label>
                    Téléphone:
                    <input type="text" readOnly={true} value={this.props.user.phone}/>
                </label>
            </div>
        )
    }
}

const jsnow = {
    lastname: 'snow',
    firstname: 'john',
    phone: '013246579'
};

ReactDOM.render(
    <UserInfos user={jsnow}/>,
    document.getElementById('root')
);
```

3 *Un peu plus de JSX*

```jsx
import React from 'react';
import ReactDOM from 'react-dom';

class ClickCounter extends React.Component {

    // initialisation de l'état
    state = {
        count: 0
    }

    incrementCounter() {
        //mise à jour de l'état, par rapport à l'état précédent
        this.setState(prevState => ({count: prevState.count + 1}))
    }

    render() {
        // Le 'text' est passé par les props alors que le nombre de click est stocké dans le state du composant
        return (
            <div>
                <button onClick={() => this.incrementCounter()}>Click me !</button>
                <span>{this.props.text}{this.state.count}</span>
            </div>
        );
    }
}

ReactDOM.render(
    <ClickCounter text="Nombre de clicks: "/>, // passage de la propriété 'text' par le composant parent
    document.getElementById('root')
);
```

6 *Utilisation des props et du state.*

```jsx
import React from 'react';
import ReactDOM from 'react-dom';

const Hello = (props) => <div>Hello {props.name}</div>; // Hello est un 'Functional Component'

ReactDOM.render(
    <Hello name="world"/>,
    document.getElementById('root')
);
```

7 *HelloWorld sous forme d'un fonctional component.*

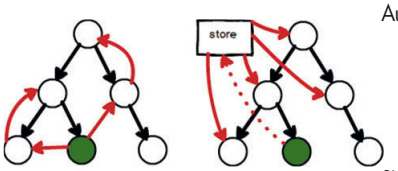
rapidement. Dans le cas classique le navigateur se connecte au serveur pour récupérer une page index.html qui va contenir des liens vers des fichiers js, et des fichiers css. Le navigateur doit donc refaire des nouvelles requêtes http vers ces fichiers. Une fois les fichiers JS récupérés, il peut les exécuter. Ces fichiers JS peuvent demander d'aller charger des contenus avec de nouvelles requêtes.C'est une approche classique mais on se rend compte que le moment où l'application va commencer à être dessinée à l'écran, et puis celui où elle est complètement affichée sont finalement assez loin dans le processus (surtout avec beaucoup de requêtes avant d'y arriver).
Générer l'application sur le serveur, permet d'envoyer du contenu dès le fichier index.html.
Évidemment il ne s'agira que d'un contenu statique qui ne peut pas encore réagir aux clics par exemple. Il n'y a pas d'autres différences avec le cas classique : dans le fichier index.html les ressources JS et CSS sont référencées et seront téléchargées. Mais en attendant que le site soit dynamique, l'utilisateur aura un premier affichage plus rapidement.
Le travail de construction sur le serveur peut être coûteux aussi en temps (il faudra aussi faire des requêtes à des ressources externes pour charger du contenu) mais on peut imaginer des stratégies de cache afin que les clients aient leur réponse le plus rapidement possible. 12

Optimisations

Chaque fois que l'état (le *state*) ou les propriétés (*props*) d'un composant changent, le composant et ses sous-composants vont être redessinés.
Le problème est que manipuler le DOM du navigateur web coûte cher, et que de supprimer un élément pour remettre exactement le même va dégrader les performances quand cela se produit sur une grande grappe d'éléments. L'idée de React est d'utiliser un *virtual DOM*, c'est à dire un DOM en mémoire. Il permet de faire une première fois les modifications, donc de façon très rapide. Une fois le *virtual DOM* mis à jour, il est comparé au véritable DOM. Si les deux sont équivalents, alors aucune modification n'est faite.
Une 2e optimisation proposée par React est de décrire dans une méthode

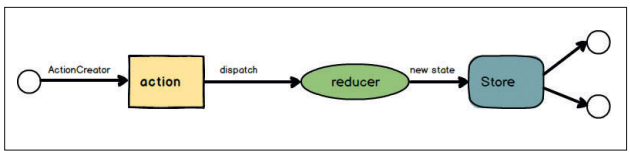
REDUX

Redux est une des librairies phares de React, qui implémente le pattern *flux* de Facebook.
L'objectif de *Redux* est de rendre un objet (le *store*) accessible à tous les composants de l'application contenant l'ensemble des propriétés que l'on veut partager. Là où les *props* permettaient de passer des valeurs à un composant enfant, le *store* permet lui de passer ces valeurs à tous les composants de l'application, ou du moins tous ceux que l'on a liés à ce store. 8



8 *Si un composant modifie une donnée, il ne doit pas avertir lui-même les autres composants. Il va modifier le store qui lui notifiera les composants que la donnée a changé.*

L'idée forte derrière *flux/redux* est que les données ne transitent que dans un seul sens. On oublie le *2-way binding* d'Angular qui crée donc un lien bidirectionnel direct entre le composant et son modèle, déclenchant la mise à jour de l'un lorsque l'autre a été modifié.
Au contraire, flux/redux prônent un flux unidirectionnel : 9 un composant graphique (un bouton sur lequel on a cliqué, un composant qui vient d'être affiché dans la page…) va créer une *action*. Une *action* est composée d'un *type* (incrémenter un compteur, ajouter un item à une todo-list, ...) et d'une *value* (la valeur de l'incrément, le texte d'une tâche à ajouter à une todo-list, ...) 10
Cette *action* doit ensuite être envoyée à Redux (on appelle ça *dispatcher* l'*action*).
L'*action* est envoyée à des *reducers*. Un *reducer* est une fonction pure qui prend deux paramètres : l'état du store de l'application (la valeur actuelle du compteur, la liste des tâches de la todo-list, …) et donc l'action qui vient d'être *dispatchée*. Le *reducer* ne modifie pas l'état si le *type* de l'action ne le concerne pas (il y a un reducer pour chaque donnée présente dans le store). Si le *type* de l'action convient alors le reducer applique une transformation (incrémenter le compteur, ajouter une tâche à une liste, …) de l'état courant avec la *valeur* passée dans l'*action*. 11
Une fois les données stockées dans le *store* mises à jour, les composants graphiques qui affichent ces données vont se voir passer (dans leurs *props*) les nouvelles valeurs. React redessine alors les composants dans le navigateur. Une nouvelle action peut être créée et envoyée à Redux !



9 *Le flux unidirectionnel, qui part d'un composant pour mettre à jour d'autres composants*

```
{type: 'INCREMENT', inc: 3}
{type: 'ADD_TODO', text: 'Acheter du pain'}
```

10 *Des Actions Redux*

```
export const display = (state = 0, action) => {
    switch (action.type) {
        case 'INCREMENT':
            return state + action.inc;
        case 'DECREMENT':
            return state - action.inc;
        default:
            return state;
    }
};
```

11 *Exemple de reducer*

Après plus de deux ans d'utilisation de Redux sur les projets, il est maintenant reproché à la librairie les différentes méthodes à écrire pour chaque donnée que l'on veut partager dans le store : une action, un reducer, et la glue pour connecter un composant au store.

(*shouldComponentUpdate*) si le composant doit être redessiné, en se basant sur les *props* ou le *state* pour prendre la décision. Attention tout de même à ce que cette méthode ne soit pas plus coûteuse en temps que la manipulation du *virtual DOM*.

React Fiber

Quand Facebook a sorti la version 16 de React en septembre 2017, c'était certes pour proposer quelques belles fonctionnalités, mais aussi pour réécrire totalement la librairie, sans changer les comportements. Pourquoi tout réécrire alors ? Afin de préparer le futur de React !

React va permettre de faire un rendu asynchrone de l'interface graphique. C'est à dire que l'affichage des composants est fait de manière non bloquante, laissant la place à des traitements plus prioritaires qui peuvent s'intercaler.

Une fois ces traitements terminés, le rendu graphique reprend là où il en était. Et si ces traitements intermédiaires ont modifié les données à afficher, seule la dernière version apparaîtra à l'utilisateur. Les états intermédiaires n'auront pas été affichés, les différents traitements étant regroupés dans un lot (*batch*) dont l'affichage marque la fin. Ceci évite de consommer des ressources à manipuler le DOM et redessiner la page dans le navigateur, alors que les éléments graphiques auraient besoin d'être mis à jour dans la fraction de seconde suivante.

Andrew Clark, l'un des développeurs de l'équipe l'explique régulièrement : il s'agit de faire de React une *Virtual Stack* dans laquelle les *fibers* (unités de traitements) sont réordonnées suivant leur priorité.

Plutôt que de rester sur le modèle classique de la pile d'appels (call stack), lorsque que des appels de fonctions s'enchaînent, React découpe ces appels en éléments distincts, appelés donc des *fibers*.

Chaque *fiber* se voit affecter une priorité suivant le type de traitement à faire : affichage, animation, calculs, interactions de l'utilisateur.

Les actions de l'utilisateur dans l'interface graphique (clics sur un bouton, saisie dans un champ texte) ou les animations seront toujours prioritaires face la mise à jour des données de l'application, donnant ainsi une impression de réactivité de l'application là où classiquement on voyait notre navigateur se figer et ne même pas inscrire le texte que l'on pouvait saisir au clavier.

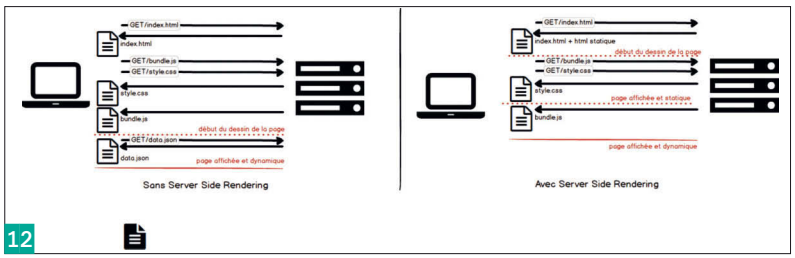
Sur une machine rapide ou avec peu de choses à afficher, il n'y a aucune différence avec un affichage synchrone, mais sur une machine plus lente ou avec trop d'éléments à redessiner, l'expérience utilisateur est nettement améliorée car il n'y a plus le sentiment d'écran gelé, qui ne répond à aucune action, comme les clics acharnés de l'utilisateur !

Le second intérêt de cet affichage asynchrone des composants est de décider quand un composant peut être affiché. C'est à dire que le développeur va pouvoir décider d'attendre que tous les appels réseaux nécessaires au composant soient terminés avant de réellement afficher le composant. Et pendant que les appels réseaux sont faits, l'application reste totalement utilisable; encore une fois les interactions auront la priorité et l'utilisateur pourra donc continuer à naviguer dans l'application même si les appels réseaux ne sont pas finis. Bien sûr, seul le dernier état de l'application sera présenté, sans l'affichage des états intermédiaires qui auront été abandonnés.

Pour le moment React n'exploite pas cette fonctionnalité, l'équipe prévoit ceci pour la fin d'année 2018 mais les démonstrations (Dan Abramov, IcelandJS 2018) permettent déjà de se rendre compte de l'intérêt de *fiber* pour améliorer l'expérience utilisateur quelle que soit la puissance de la machine, et la qualité du réseau.

React affiche de belles promesses pour se démarquer de ses concurrents, la fin d'année 2018 va être très excitante !

Le mois prochain, nous verrons dans la partie 2, des cas pratiques React ! •



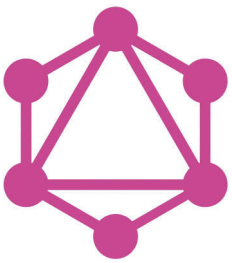
12

IMMUTABILITÉ

Dans des cas simples comme la gestion de l'état d'un composant, ou encore dans la gestion du *store Redux*, React prône l'immutabilité des objets et l'utilisation des fonctions pures.

Pour modifier la propriété d'un objet, il faut en créer un nouveau. Outre la lisibilité du code : les résultats de fonctions ne dépendent que des paramètres en entrée, les fonctions n'ont pas d'effet de bord, ... Cela permet surtout de comparer très rapidement l'égalité entre 2 objets : soit ils ont la même référence en mémoire (égalité stricte ===) et c'est donc le même objet, soit ils sont différents. C'est plus rapide que de comparer les propriétés une à une !

GRAPHQL



Si *Redux* s'est imposé sur de nombreux projets React, une autre librairie qui lui était avant complémentaire, tend aujourd'hui à le remplacer sur les nouveaux projets. Il s'agit de Graphql.

Graphql est un outil permettant de réduire le nombre de requêtes au backend ainsi que de ne récupérer que le contenu nécessaire dans les réponses.

Cela s'oppose à une stratégie d'API REST orientée ressources, sur laquelle on va peut-être multiplier les appels afin d'obtenir toutes les informations à afficher sur une page et qui parfois renvoie plus d'informations que nécessaire dans les réponses.

Il faut donc écrire un client qui décrit les requêtes à faire dans un appel, ainsi que les éléments attendus dans la réponse.

Par exemple dans le cas d'un agenda, le serveur peut proposer de récupérer la liste des rendez-vous avec pour chacun les personnes rencontrées et des informations sur elles, le lieu et l'heure.

Là où en REST, on aurait d'abord demandé la liste des rendez-vous puis dans une seconde requête les détails sur chaque invité (noms, téléphones, mails, ...), en *Graphql* avec une seule requête on va demander la même liste de rendez-vous, avec pour chacun d'eux juste les noms des personnes présentes. On s'économise donc un appel, et surtout la latence réseau qui lui est inhérente. En plus le poids de la réponse est plus faible, étant donné que certaines infos inutiles ne sont pas envoyées (les téléphones et mails).

Sur la partie serveur de *Graphql*, il faut décrire les types des objets renvoyés et comment chaque type d'information va être cherché : dans une base de données, sur une API REST, ...

Le client *Graphql* permet aussi de mettre en cache le résultat des requêtes.

Il est aussi possible de s'en servir comme d'un *store* où sont stockées les données transverses de l'application.

Graphql est donc en concurrence directe avec Redux sur certains points, et complémentaire sur d'autres.

Philippe Charrière
@k33g_org
Technical Account Manager pour GitLab
Associé chez Clever Cloud
Fondateur de Bots.Garden (éleveur de bots)

niveau 200

Prise en main de GitLab en douceur

Partie 2

Code, test, and deploy together

FAISONS UN PEU D'INTÉGRATION CONTINUE

En ce qui concerne l'intégration continue **GitLab** propose le "tooling" nécessaire (même dans la version communauté) avec les **GitLab Runners**.

Mais qu'est-ce qu'un GitLab Runner?

Un runner (au sens **GitLab**) est un exécutable, qui, à partir d'un fichier présent dans votre projet, va déclencher différentes actions en fonction des évènements qui arrivent sur votre projet (par exemple, à chaque commit sur une branche particulière, déclencher des tests unitaires).

Cet exécutable permet de créer différents types de runner (on parlera d'**executor**) (ils sont aussi multi-plateformes): shell, Docker, Docker-Machine (auto-scaling), Parallels, VirtualBox, SSH, Kubernetes.

Pour commencer je vais créer un runner de type **shell** dans une VM à partir de **Vagrant**. Je vais avoir besoin d'un executor et de nodejs.

Dans un premier temps, nous avons besoin d'un token

Dans l'interface web de votre instance **GitLab**, allez dans les **Settings** de votre projet, puis sélectionnez le sous-menu **CI/CD** : 23

Puis cliquez sur le bouton **Expand** de la zone **Runners settings**:

Vous allez obtenir les informations nécessaires pour connecter votre runner à votre projet (pour information les runners peuvent être spécifiques à un projet, ou un groupe, ou une instance **GitLab**, dans les 2 derniers cas on parle de runners partagés) : 24

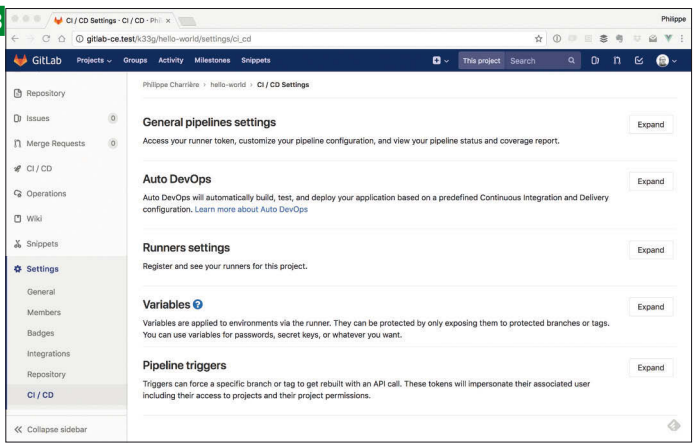
Vous remarquerez dans la zone **Setup a specific Runner manually** que vous avez l'url de votre instance **GirLab** et un **registration token**. A partir de ces informations, nous allons pouvoir créer notre runner.

Création et lancement du runner

Voici donc mon Vagrantfile (à mettre dans un répertoire runner par exemple) :

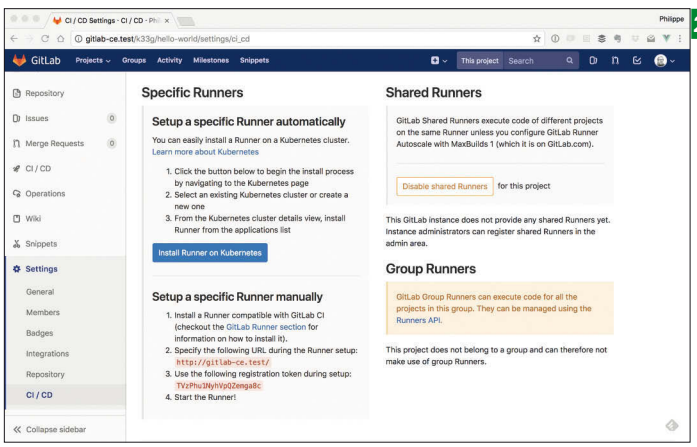
```
CI_REGISTRATION_TOKEN="TVzPhu1NyhVpQZemga8c"

GITLAB_DOMAIN = "gitlab-ce.test"
GITLAB_IP = "172.16.245.121"

SHELL_TESTS_RUNNER_NODE_NAME="testnoderunner"
SHELL_TESTS_RUNNER_NODE_IP="172.16.245.131"

BOX_IMAGE = "bento/ubuntu-16.04"
VERSION = "1.0"

Vagrant.configure("2") do |config|
 config.vm.box = BOX_IMAGE

 ENV['LC_ALL']="en_US.UTF-8"

 config.vm.define "#{SHELL_TESTS_RUNNER_NODE_NAME}" do |node|

  node.vm.hostname = "#{SHELL_TESTS_RUNNER_NODE_NAME}"

  node.vm.network "public_network", bridge: "en0: WiFi (AirPort)"
  node.vm.network "private_network", ip: "#{SHELL_TESTS_RUNNER_NODE_IP}"

  node.vm.provider "virtualbox" do |vb|
   vb.memory = 512
   vb.cpus = 1
   vb.name = "#{SHELL_TESTS_RUNNER_NODE_NAME}"
  end

  node.vm.provision :shell, inline: <<-SHELL
```

23



Settings > CI/CD

24



Specific Runners

```
  # (1) Install GitLab Runner
  curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh
| sudo bash
  apt-get install -y gitlab-runner

  # (2) Install NodeJS
  curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
  apt-get -y install nodejs

  # (3) Add entries to hosts file
  echo "" >> /,etc./hosts
  echo '#{GITLAB_IP} #{GITLAB_DOMAIN}' >> /,etc./hosts
  echo "" >> /,etc./hosts

  # (4) Registering GitLab Runner
  gitlab-runner register —non-interactive \
    —url "http://#{GITLAB_DOMAIN}/" \
    —name "shell_runner_tests_nodejs" \
    —registration-token #{CI_REGISTRATION_TOKEN} \
    —tag-list "test" \
    —executor shell

  SHELL

 end

end
```

Quelques explications :

- **(1) Install GitLab Runner** : je télécharge le package **GitLab Runner** et je l'installe ;
- **(2) Install NodeJS** : je télécharge le package **Node JS** et je l'installe (***pour rappel, mon application est une application node***) ;
- **(3) Add entries to hosts file** : la VM de mon runner a besoin de pouvoir "voir" mon instance **GitLab** sur le réseau, donc je vais lui ajouter l'ip et le nom de domaine de l'instance dans son fichier /,etc./hosts ;
- **(4) Registering GitLab Runner** : et enfin j'enregistre mon runner auprès de mon instance **GitLab** en lui précisant son token avec —registration-token #{CI_REGISTRATION_TOKEN}, son type avec —executor shell et je lui donne un tag: —tag-list "test".

Nous le verrons plus tard, mais le **tag** d'un runner est important, cela va vous permettre de choisir les runners auxquels vous allez déléguer des tâches (tests, builds, déploiement, qualité de code, …) dans le cas où vous utilisez plusieurs runners pour un projet.

Lancement du runner

Lancez la création de la VM et l'exécution du runner avec la commande `varan up`.

Une fois votre runner initialisé et démarré, vous pouvez aller vérifier qu'il a bien été enregistré auprès de l'instance **GitLab**. Retournez dans le menu **Settings** de votre projet, sélectionnez le sous-menu **CI/CD** puis le bouton **Expand** de la section **Runners settings**, ainsi vous allez pouvoir vérifier le bon enregistrement du runner : 25

Vous avez donc bien un runner nommé `shell_runner_tests_nodejs` et "tagué" `test`.

Jouons avec le runner

Il est donc temps de jouer un peu avec notre runner. Nous allons ajouter dans notre projet le fichier magique qui va nous permettre de déclencher des actions à faire exécuter par notre runner. Ce fichier s'appelle .gitlab-ci.yml et il va contenir tous les ordres à passer au runner.

Comme c'est un tutorial, nous allons nous permettre quelques libertés : nous allons ajouter et modifier des fichiers directement à partir de l'interface web de notre instance **GitLab** plutôt que de faire ça sur notre poste, puis de commiter et pousser nos modifications (***bien sûr, vous n'oublierez pas à la fin de re synchroniser votre projet***). C'est parti !

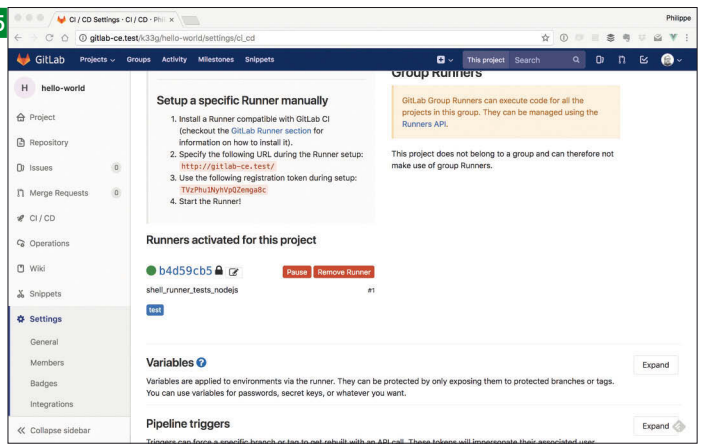
Retournez à la racine du projet dans l'interface web, et déroulez la zone de liste "+" et sélectionnez l'option de menu **New file** : 26

Et créez un fichier nommé .gitlab-ci.yml avec le contenu suivant : 27

Le code complet : 28

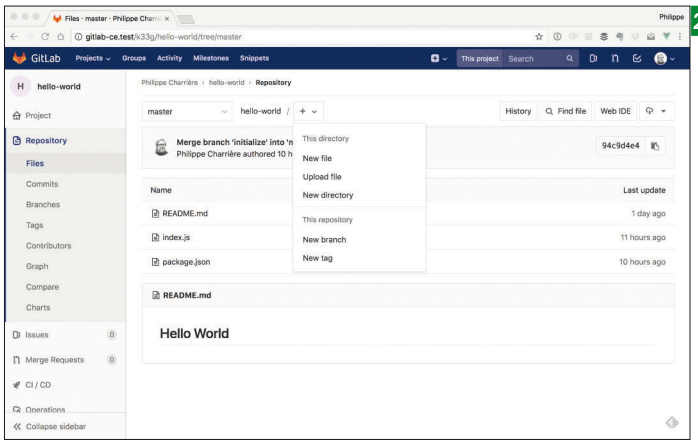
```
# my first CI file
stages:
 - one
 - two

only_feature_branches:
 stage: one
 except:
```

25



Runner settings

26



Ajouter un fichier

```
 - master
script:
 - echo "  some stuff on $CI_COMMIT_REF_NAME"

only_master:
 stage: two
 only:
  - master
 script:
  - echo "  some stuff on master"
```

Nous avons défini 2 **stages** (des étapes) one et two, ensuite nous avons défini 2 **jobs** only_feature_branches et only_master. Un job est un ensemble de commandes que l'on veut faire exécuter à notre runner sous certaines conditions.

Par exemple pour only_feature_branches, nous souhaitons :

- exécuter la commande echo " some stuff on $CI_COMMIT_REF_NAME" grâce au mot-clé script (la variable $CI_COMMIT_REF_NAME permettra d'afficher le nom de la branche impactée),
- ne le faire que pour les branches qui ne sont pas master grâce au mot-clé clé except avec la valeur master,
- et tout cela appartient à l'étape one (stage: one).

Ensuite, commitez vos modifications directement sur la branche master :

Si vous allez dans le menu **CI/CD** (celui avec la petite fusée) et que vous sélectionnez le sous-menu **Pipelines**, vous obtenez la liste des pipelines en cours, passés ou à venir. Voyez un pipeline comme une suite d'actions à exécuter pour votre projet. Dans notre cas, vous pouvez noter qu'un pipeline a été créé (car nous avons commité des changements), mais qu'il a un statut **pending** et qu'il a hérité d'un tag **stuck** : 29

Ce tag **stuck** signifie que notre pipeline n'a pas réussi à trouver un runner pour s'exécuter. Si vous vous souvenez, tout à l'heure je vous ai expliqué que les tags des runners étaient important pour permettre de dispatcher le travail sur différents runners. En fait nous devons préciser pour chaque job quel type de runner utiliser.

Donc, retournez éditer le fichier .gitlab-ci.yml et précisez pour chacun des jobs qu'ils doivent utiliser les runners tagués **test** avec le mot-clé tags : 30

Donc, ajoutez juste ceci :

```
tags:
 - test
```

Commitez vos changements.

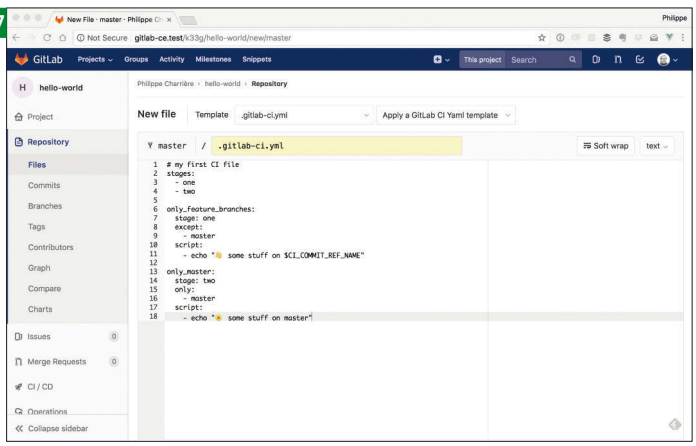
Si vous retournez sur la liste des pipelines, vous pouvez voir que cette fois votre pipeline s'est bien exécuté : 31

Si vous cliquez sur le **#2** (l'identifiant du pipeline), vous allez arriver sur la représentation graphique (très simple dans notre cas) de votre pipeline : 32

Donc vous voyez que votre étape **only_master** est au vert (tous les jobs se sont bien passés), cliquez sur le bouton associé à votre étape **only_master** et vous allez pouvoir découvrir les résultats de l'exécution de votre pipeline : 33

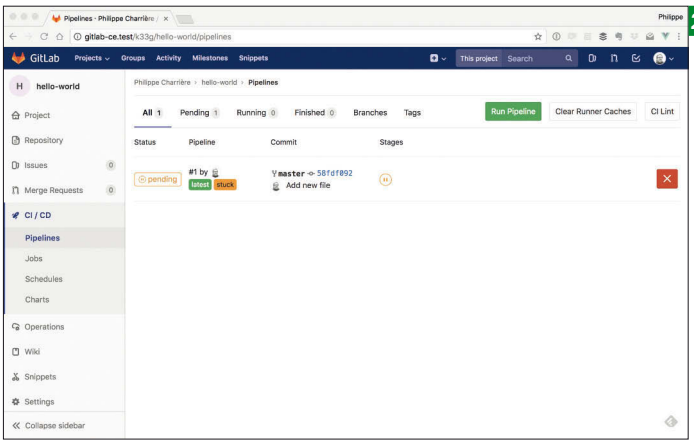
Donc vous pouvez voir que le runner a cloné votre repository et exé-

27



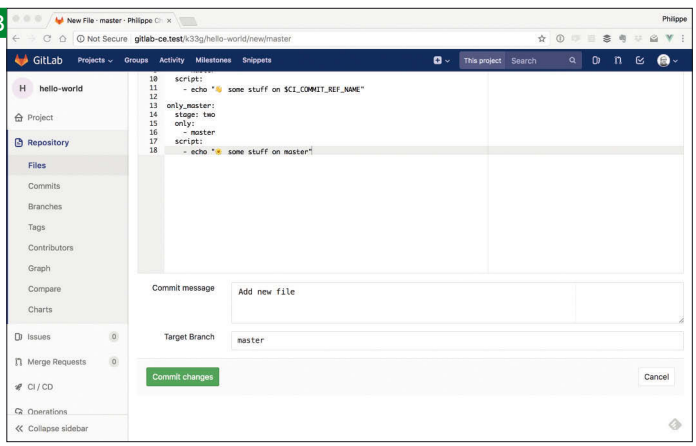
.gitlab-ci.yml

29

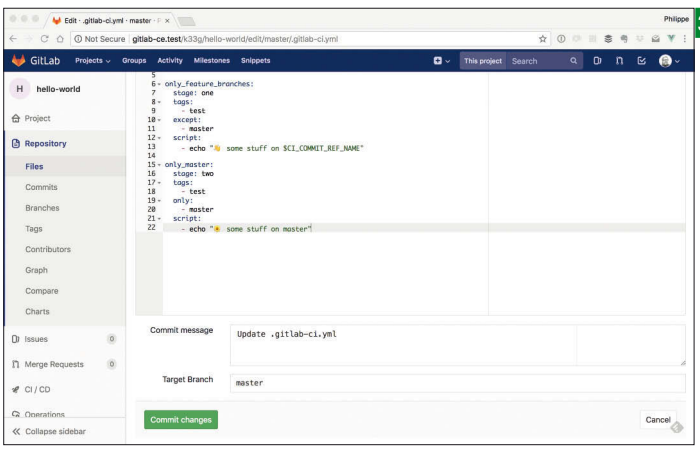


Commit changes

28



30



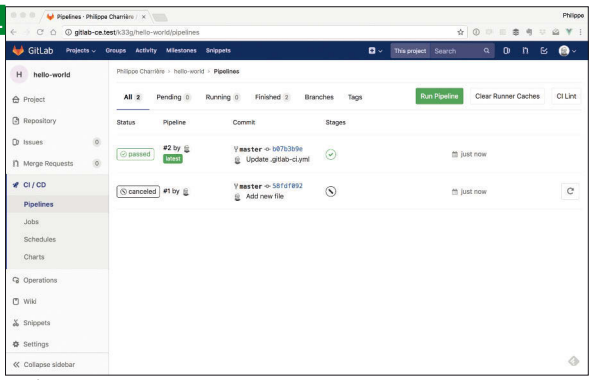
Commit changes

cuté votre commande `echo`. Jusqu'ici c'est simple. Retournons modifier notre fichier .gitlab-ci.yml: .
Et changeons le 2eme job par :

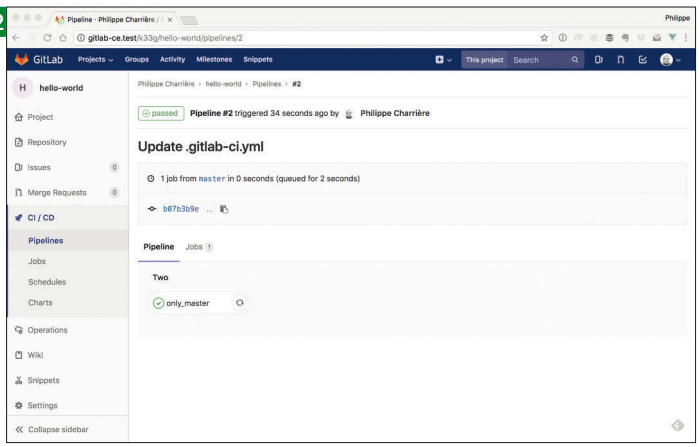
```
not_only_master:
 stage: two
 tags:
  - test
 script:
  - echo " some stuff on all branches"
```

Ensuite, allez modifier (toujours directement dans l'interface web) le fichier README.md .
Et commitez directement vos changements sur une nouvelle branche update-readme (il suffit de saisir son nom dans le champ text Target branch pour créer cette nouvelle branche à partir de master) 34.
Une fois que vous avez cliqué sur le bouton **Commit changes** vous arrivez sur l'écran de création de **Merge Request**, donc soumettez votre **MR**.
Vous arrivez ensuite sur l'écran de suivi de la **Merge Request** et vous pouvez noter qu'un nouveau pipeline s'est déclenché et que toutes ses étapes (2 dans notre cas) sont au vert (nos traitements sont courts donc quasi instantanés, c'est pour cela que nous avons directement les résultats des pipelines dans la fiche de la **MR**) : 35
Cliquez sur le **#6** à côté du mot **Pipeline** (selon ce que vous avez fait, bien sûr ce numéro peut être différent), et cela va vous amener sur la représentation graphique de votre pipeline.
Vous notez cette fois-ci que 2 étapes s'affichent. Cliquez sur l'étape only_feature_branches et vous voyez le résultat et pouvez vérifier que le runner a bien utilisé la variable $CI_COMMIT_REF_NAME pour afficher le nom de la branche en cours update-readme
Donc, vous voyez que l'utilisation et le "pilotage" des runners sont plutôt simples (bien sûr, il est tout à fait possible de faire des choses beaucoup plus complexes, mais ce sera pour une autre fois). Il est maintenant temps d'ajouter des tests automatiques à notre chaîne de CI. Avant cela, retournez sur votre **MR**, dans la barre de menu de gauche, sélectionnez le menu **Merge Requests**, sélectionnez votre **MR** et mergez-la à master.
Vous pouvez vérifier que cela a déclenché un nouveau pipeline.

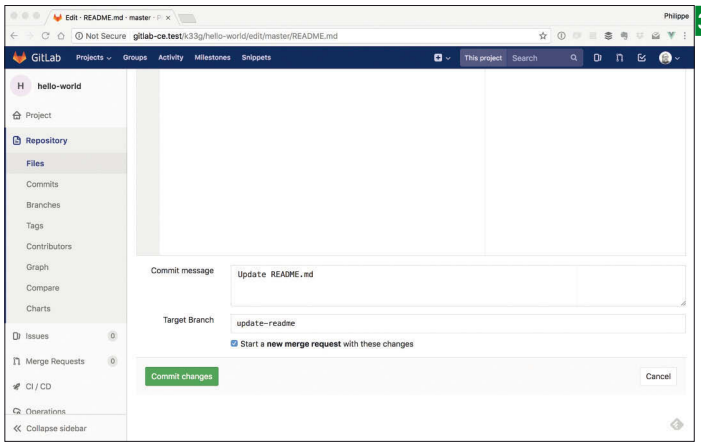
La suite de l'article le mois prochain

31


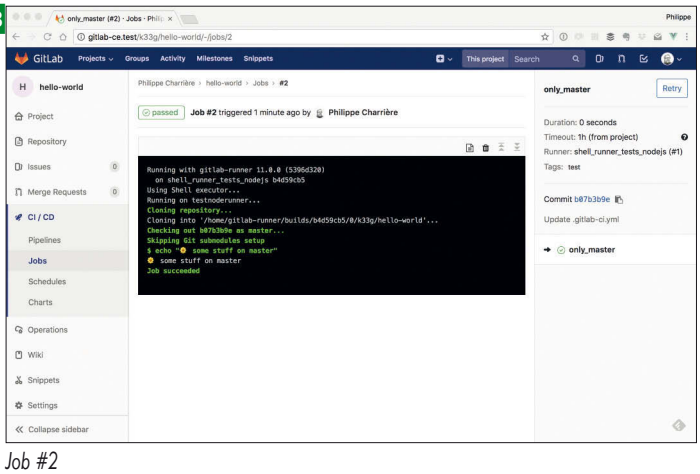
Pipelines

32


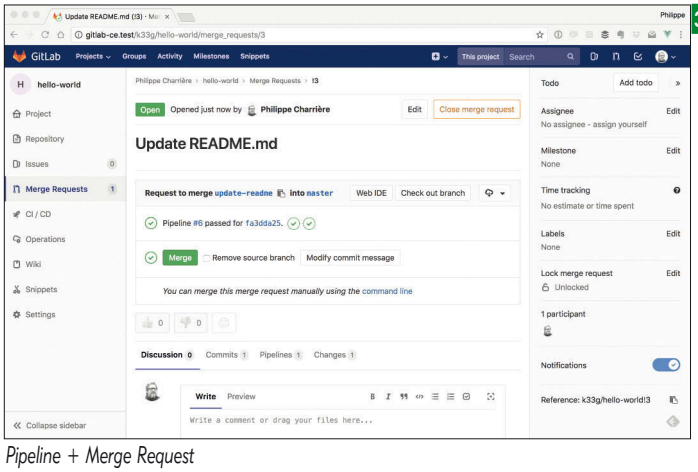
Pipeline only_master

34


Commit changes

33


Job #2

35


Pipeline + Merge Request

Christophe PICHAUD
Consultant sur les technologies Microsoft
christophepichaud@hotmail.com |
www.windowscpp.com

Linux : le multithreading en C++

niveau 200

Dans cet article, nous allons parler de la plateforme Linux. Nous allons expliciter ce qu'est du multithreading et comment en faire en C++ sous Linux. Pour cela nous allons prendre comme support la STL (Standard Template Library) du C++ qui fournit tout ce qu'il faut.

Un environnement intégré : VSCode

Microsoft fournit un IDE gratuit et léger qui est VSCode. Cet IDE est disponible sous Windows, Mac et Linux. Mon poste est sous Ubuntu et VSCode est sympa à utiliser. 1

Comme vous pouvez le voir sur la photo d'écran, VSCode contient un terminal pour lancer la compilation.

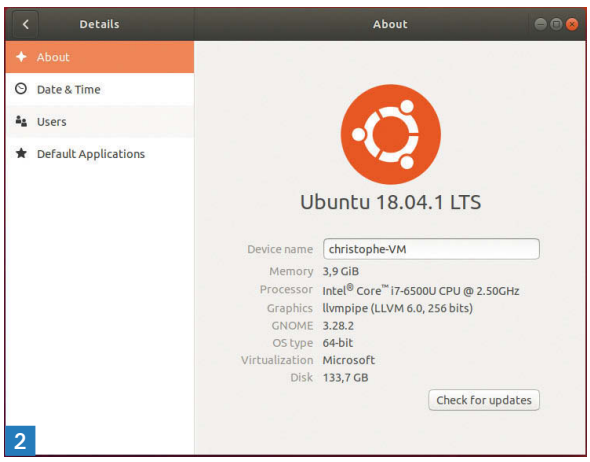
Linux

Mon environnement est un Ubuntu Desktop 18.04. 2

Maintenant que nous avons un système Linux et un IDE, il nous reste à choisir un compilateur : GCC. C'est le standard historique. On ne change pas une équipe qui gagne ! Avec un g++ –v, je suis en version GCC 7.3. Il supporte le C++11 et C++14 et C++17 : c'est l'essentiel.

Le multithreading et les threads, c'est quoi ?

Un thread est une routine qui s'exécute en arrière-plan. C'est le système d'exploitation qui gère les threads. Sous Windows, le système d'exploitation ordonnance des threads, sous Linux, le système d'exploitation ordonnance des processus. Le thread est un mécanisme léger. Le thread permet d'accéder à des ressources ; pour éviter les conflits d'accès, il faut protéger ces ressources. Exemple : si une variable globale est utilisée dans deux threads qui s'exécutent, il peut y avoir une corruption de mémoire. Le premier thread fait une opération d'affectation, il est préempté (il devient inactif) et le deuxième thread devient actif et modifie à son tour la variable. Dans ce cas précis, il y a corruption de mémoire car l'emplacement mémoire est modifié au même moment... Pour éviter cela, on va mettre un blocker de type mutex, semaphore ou barrier.



2

La STL ou les API systèmes ?

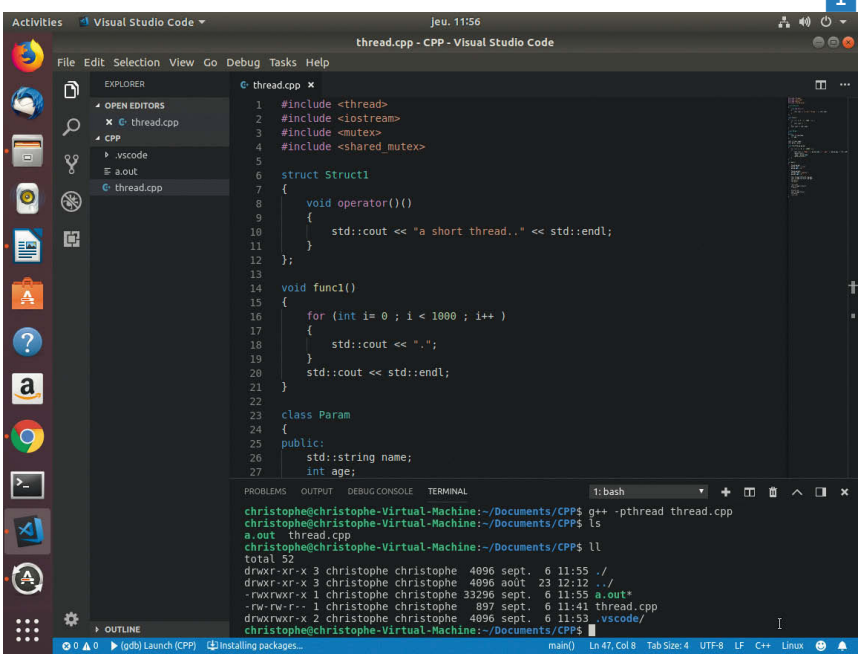
Pour créer un thread, on a deux solutions. Soit on utilise les API du système d'exploitation donc dans notre cas la librairie pthread, soit on utilise la STL et son fichier d'entête <thread>. L'avantage d'utiliser la STL est que le code est portable ou cross-plateforme. En utilisant directement pthread, mon code ne tournera que sur Linux… C'est la même chose sous Windows, si j'utilise la routine Win32 CreateThread, mon code est verrouillé pour Windows…

Mon premier thread

Le fichier d'entête <thread> de la STL permet de gérer les threads. Un thread est créé en instanciant la classe thread, et en lui passant en paramètre, soit une structure, une classe ou une fonction. La méthode join() permet d'attendre la fin d'exécution du thread.

```
#include <thread>
#include <iostream>

struct Struct1
{
   void operator()()
   {
      std::cout << "a short thread.." << std::endl;
```

1



```
    }
};

void func1()
{
    for (int i= 0 ; i < 1000 ; i++ )
    {
        std::cout << ".";
    }
    std::cout << std::endl;
}

int main()
{
    std::thread t(func1);
    t.join();

    Struct1 s1;
    std::thread t2(s1);
    t2.join();
   return 0;
}
```

On remarquera que pour être utilisé avec une structure ou une classe, un thread utilise la redéfinition de l'opérateur () pour s'exécuter. La forme la plus simple est sinon de fournir une simple routine. Les threads s'exécutent dans le même espace mémoire et peuvent communiquer au travers d'objets partagés. De telles communications sont protégées par des verrous pour éviter les concurrences d'accès qui provoquent des plantages d'application car il y a corruption de la mémoire en cas d'accès concurrents sur la même donnée.

Compilation avec g++

La suite de compilation est celle de GCC et plus particulièrement g++ qui est le compilateur c++. Pour compiler un programme qui utilise les threads, il faut préciser l'option –pthread ; on utilise le standard C++17 via –std:c++17 :

```
$ g++ -pthread –std=c++17 thread.cpp –o thread
```

Cela produit un fichier exécutable thread que l'on peut lancer via ./thread.

Passage d'arguments

Pour pouvoir effectuer son job, un thread a besoin de paramètre(s). Il suffit de les passer en arguments au constructeur de la classe thread :

```
class Param
{
public:
    std::string name;
    int age;
};

void func3(Param param)
{
    std::cout << "Name:" << param.name << ", Age:" << param.age << std::endl;
}

int main()
{
    Param param;
    param.name = "Lisa";
    param.age = 12;
    std::thread t3(func3, param);
    t3.join();
}
```

Dans cet exemple, la routine func3 prend une classe Param en paramètre.

Retour de valeur

Il est possible pour un thread de retourner des éléments, une valeur, des champs, etc. **Voir code complet sur** www.programmez.com Dans cet exemple, on passe l'adresse d'un double au constructeur de la structure en plus de la classe Param. On fait pointer ce membre sur l'adresse d'une variable membre et ensuite, le thread modifie la valeur membre. C'est tout !

Partage de données

Il est parfois nécessaire de partager des données. Dans ce cas, les accès aux données doivent être protégés. L'objet le plus simple pour y parvenir est le mutex : *mutual exclusion object*. Cet objet est généralement déclaré en global ou dans la classe et est utilisé pour protéger un bloc de données.

```
std::string _name;
std::mutex _mutex;

void func5(Param param)
{
    for (int i = 0; i < 100; i++)
    {
        std::cout << "Name:" << param.name << ", Age:"
<< param.age << std::endl;
        _mutex.lock();
        _name = param.name;
        _mutex.unlock();
    }
}

int main()
{
    Param param;
    param.name = "Lisa";
    param.age = 12;

    Param param2;
    param2.name = "Audrey";
    param2.age = 8;
```

```
    std::thread t5(func5, param);
    std::thread t6(func5, param2);
    t5.join();
    t6.join();
}
```

La meilleure façon d'introduire une pause dans un thread est d'utiliser la routine sleep_for() elle redonne la main aux autres threads. Voici comment faire une attente de 250 millisecondes :

```
std::this_thread::sleep_for(std::chrono::milliseconds(250));
```

Il est aussi possible d'utiliser un lock plus léger qui est le shared_mutex :

```
std::shared_mutex _smutex;

void func6(Param param)
{
    for (int i = 0; i < 10000; i++)
    {
        _smutex.lock();
        _name = param.name;
        _smutex.unlock();

        _smutex.lock_shared();
        std::cout << "Name:" << _name << std::endl;
        _smutex.unlock_shared();
    }
}

int main()
{
    Param param;
    param.name = "Lisa";
    param.age = 12;

    Param param2;
    param2.name = "Audrey";
    param2.age = 8;

    std::thread t5(func6, param);
    std::thread t6(func6, param2);
    t5.join();
    t6.join();
```

Dans ce cas, on utilise la méthode lock_shared() pour l'accès en lecture et la méthode lock() pour l'accès en écriture.

Tâches : future et promise

Il existe un mécanisme de plus haut niveau que les threads et les mutex : ce sont les classes disponibles dans l'entête <future>. Lorsqu'un transfert de valeur entre deux tâches peut se faire sans verrous, on met en œuvre les futures et les promises, le tout étant géré par des tâches plutôt que des threads. Quand une tâche veut passer une valeur à une autre tâche, elle positionne la valeur dans la promise ; la mécanique fait que la valeur est accessible en lecture dans la future. L'appel se fait de la sorte si nous avons une future<X> fx:

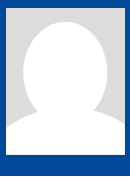
```
X x = fx.get();

Exemple de future :
int MySum(int i, int j)
{
    return i + j;
}

void ft()
{
    std::packaged_task<int(int, int)> task(MySum); // wrap the function
    std::future<int> f1 = task.get_future();  // get a future
    std::thread t(std::move(task), 10, 20); // launch on a thread
    f1.wait();
    std::cout << "Done! result: " << f1.get() << std::endl;
    t.join();
}
```

Il est aussi possible de faire ça avec une promise en mettant la valeur via un set_value :

```
void ft3()
{
    // future from a promise
    std::promise<int> p;
    std::future<int> f3 = p.get_future();
    std::thread([&p] { p.set_value_at_thread_exit(9); }).detach();

    std::cout << "Waiting..." << std::flush;
    f3.wait();
    std::cout << "Done! result: " << f3.get() << std::endl;
}

Le plus simple restant l'utilisation de async :
void ft2()
{
    // future from an async()
    std::future<int> f2 = std::async(std::launch::async, [] { return 8; });
    f2.wait();
    std::cout << "Done! result: " << f2.get() << std::endl;
}
```

Conclusion

L'utilisation des threads n'est pas très compliquée. Il suffit de prendre garde à bien protéger les données partagées et de mettre en place les mutex qui vont bien. Pour une utilisation de plus haut-niveau, on notera que async peut-être facile à utiliser aussi. Le C++ est ISO et multiplateforme donc ce qui est valable sur Linux l'est aussi sur les autres plateformes comme Android, iOS, macOS ou Windows via la STL (Standard Template Library) du C++. •



Sylvain SAUREL
Développeur Java / Android
https://www.ssaurel.com - sylvain.saurel@gmail.com

Création d'une Blockchain en Java : gestion des transactions

niveau 200

Partie 2

Au coeur du Bitcoin et des crypto monnaies, la technologie Blockchain est une véritable révolution technologique. Nous vous avions ainsi présenté dans le numéro 216 de Programmez comment réaliser les fondations de votre propre Blockchain en Java. Dans cet article, nous passons aux choses sérieuses puisque nous allons y ajouter la gestion des transactions.

Dans la première partie de cet article, publiée dans le numéro 216 de Programmez, vous avez découvert comment créer une Blockchain basique en Java. La Blockchain ainsi créée était parfaitement fonctionnelle mais se limitait au stockage de messages en mode local. La suite logique est de rajouter une gestion des transactions entre utilisateurs. Nous allons donc remplacer les messages par des transactions au sein des blocs de notre Blockchain. A la fin de cet article, nous aurons ainsi implémenté le fonctionnement d'une crypto monnaie basique.

Création d'un Portefeuille

Dans le petit monde des crypto monnaies, la propriété d'une monnaie est transférée sur la Blockchain via des transactions. Les participants à une transaction ont une adresse leur permettant de recevoir ou d'envoyer des fonds. De manière basique, un portefeuille de crypto monnaies va stocker ces adresses. Cependant, la plupart des portefeuilles actuels ont des fonctionnalités étendues puisqu'ils sont capables de créer de nouvelles transactions sur la Blockchain.

Partant de ce constant, nous allons créer un objet *Wallet* avec les 2 propriétés suivantes :

- *publicKey* de type *PublicKey* qui joue le rôle d'adresse et qui permet de recevoir des fonds sur le portefeuille.
- *privateKey* de type *PrivateKey* qui sera utilisée pour signer les transactions lors des envois de fonds. Cette clé doit absolument être gardée secrète.

Cela nous donne le corps suivant pour notre classe *Wallet* :

```
import java.security.PrivateKey;
import java.security.PublicKey;

public class Wallet {

  public PrivateKey privateKey;
  public PublicKey publicKey;

}
```

Génération des clés

Pour chaque instance de portefeuille, il est nécessaire de générer une paire de clés publique et privée. Comme expliqué précédemment, la clé privée doit rester secrète. En revanche, la clé publique est envoyée lors d'une transaction et permet de vérifier que la signature d'une transaction est bien valide. De fait, on s'assure que ses données n'ont pas été corrompues. Pour la génération des clés, nous allons nous appuyer sur l'API *KeyPairGenerator* fournie en standard dans le JDK et implémentant une méthode cryptographique basée sur les courbes elliptiques. Nous ajoutons ainsi la méthode *generateKeyPair* à notre objet *Wallet* :

```
public void generateKeyPair() {
 try {
  KeyPairGenerator keyGen = KeyPairGenerator.getInstance("ECDSA", "BC");
  SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
  ECGenParameterSpec ecSpec = new ECGenParameterSpec("prime192v1");
  keyGen.initialize(ecSpec, random);
  KeyPair keyPair = keyGen.generateKeyPair();
  // on récupère les clés générées pour notre Wallet
  privateKey = keyPair.getPrivate();
  publicKey = keyPair.getPublic();
 } catch (Exception e) {
  throw new RuntimeException(e);
 }
}
```

Les clés générées pour notre portefeuille, nous allons pouvoir les utiliser pour signer les transactions.

Modélisation d'une Transaction

Au sein de la Blockchain, le transfert de propriété d'une crypto monnaie se fait via une transaction. Chaque transaction doit contenir les données suivantes :

- La clé publique de l'expéditeur des fonds qui fait office d'adresse d'expédition pour la transaction ;
- La clé publique du destinataire des fonds qui fait office d'adresse de destination pour la transaction ;
- Le montant des fonds à transférer ;
- Une liste des précédentes transactions reçues prouvant que l'expéditeur a la propriété des fonds qu'il cherche à envoyer ;
- Une liste des précédentes transactions envoyées ;
- Une signature cryptographique prouvant que le propriétaire de l'adresse est bien celui qui réalise la transaction et envoie les fonds. La signature garantit en outre que les données de la transaction sont bien valides à la réception.

On crée donc une classe *Transaction* avec le corps suivant :

```
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.ArrayList;

public class Transaction {

  public String transactionId;
  public PublicKey sender;
  public PublicKey recipient;
  public float value;
  public byte[] signature;
  public ArrayList<TransactionInput> inputs = new ArrayList<TransactionInput>();
  public ArrayList<TransactionOutput> outputs = new ArrayList<TransactionOutput>();
  // Nombre de transactions générées
  private static int sequence = 0;

  public Transaction(PublicKey from, PublicKey to, float value, ArrayList<TransactionIn
put> inputs) {
    this.sender = from;
    this.recipient = to;
    this.value = value;
    this.inputs = inputs;
  }

  // ...
}
```

Les classes *TransactionInput* et *TransactionOutput* seront détaillées plus tard.

Gestion des signatures

Les signatures cryptographiques vont jouer 2 rôles importants dans notre Blockchain. En premier lieu, elles vont permettre aux possesseurs de notre crypto monnaie de dépenser leurs fonds. En second lieu, elles vont assurer l'intégrité des données des transactions et éviter qu'une personne malveillante tente de modifier les données d'un bloc qui va être miné.

Nous nous appuyons sur 2 clés : une publique et une privée. La clé privée sera utilisée pour signer les données de la transaction et devra absolument demeurer secrète. La clé publique, servant d'adresse, peut être également utilisée pour vérifier son intégrité. Nous allons donc devoir créer un certain nombre de méthodes utilitaires dans notre projet afin de signer une donnée au format chaîne de caractères avec la clé privée mais également afin de vérifier la signature des données avec la clé publique :

```
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;

public class Utils {

 // Application d'une signature avec les courbes elliptiques (ECDSA)
 public static byte[] applyECDSASig(PrivateKey privateKey, String input) {
  Signature dsa;
  byte[] output = new byte[0];

  try {
   dsa = Signature.getInstance("ECDSA", "BC");
   dsa.initSign(privateKey);
   byte[] strByte = input.getBytes();
   dsa.update(strByte);
   byte[] realSig = dsa.sign();
   output = realSig;
  } catch (Exception e) {
   throw new RuntimeException(e);
  }

  return output;
 }

 // Vérification de la signature pour les données d'entrée
 public static boolean verifyECDSASig(PublicKey publicKey, String data, byte[] signature) {
  try {
   Signature ecdsaVerify = Signature.getInstance("ECDSA", "BC");
   ecdsaVerify.initVerify(publicKey);
   ecdsaVerify.update(data.getBytes());

   return ecdsaVerify.verify(signature);
  } catch (Exception e) {
   throw new RuntimeException(e);
  }
 }

 // ...
}
```

Nos méthodes utilitaires créées, nous allons les utiliser dans notre classe *Transaction* afin de générer la signature d'une transaction et de vérifier cette signature, ce qui nous garantira l'intégrité des données qu'elle contient. On doit également créer une méthode *calculateHash* générant le hash des données de la transaction en y appliquant l'algorithme SHA-256 comme nous l'avions fait sur les données d'un bloc dans la première partie de cet article.

On doit donc ajouter le code suivant à notre classe *Transaction* :

```
public class Transaction {

 // ...

 private String calculateHash() {
  sequence++;
  return Utils.applySha256(Utils.getStringFromKey(sender) +
          Utils.getStringFromKey(recipient) +
            Float.toString(value) + sequence);
 }

 public void generateSignature(PrivateKey privateKey) {
  String data = Utils.getStringFromKey(sender)
          + Utils.getStringFromKey(recipient)
```

```
                + Float.toString(value);
  signature = Utils.applyECDSASig(privateKey, data);
 }

 public boolean verifiySignature() {
  String data = Utils.getStringFromKey(sender)
                + Utils.getStringFromKey(recipient)
                + Float.toString(value);
  return Utils.verifyECDSASig(sender, data, signature);
 }

 // ...
}
```

La vérification de la signature d'une transaction sur notre Blockchain sera faite par les mineurs au moment où ils vont tenter d'ajouter une transaction à un nouveau bloc.

Gestion des envois et des réceptions

Pour les débutants en crypto monnaies, la suite va être particulièrement intéressante. Afin que vous puissiez posséder 1 Bitcoin par exemple, vous devez recevoir 1 Bitcoin. Au sein du grand livre de comptes de la Blockchain, on ne va pas vous ajouter 1 Bitcoin et retirer 1 Bitcoin à l'expéditeur. A la place, l'expéditeur a gardé la référence du fait qu'il avait précédemment reçu 1 Bitcoin. On va donc pouvoir créer une transaction sortante indiquant que 1 Bitcoin a été envoyé à votre adresse. Ainsi, la liste des transactions entrantes référencent les transactions précédemment sortantes.
Au final, le montant contenu dans un portefeuille va donc être la somme des transactions sortantes non dépensées qui y ont été adressées. Au sein de notre classe *TransactionInput*, on va donc avoir un id référençant la transaction sortante associée ainsi qu'un objet de type *TransactionOutput* qui contient la transaction sortante non dépensée. En suivant les conventions de nommage de la Blockchain Bitcoin, on va nommer cette dernière propriété *UTXO*. Ceci nous donne le code suivant pour la classe TransactionInput :

```
public class TransactionInput {

 public String transactionOutputId;
 public TransactionOutput UTXO;

 public TransactionInput(String transactionOutputId) {
  this.transactionOutputId = transactionOutputId;
 }
}
```

La classe *TransactionOutput* modélise une transaction sortante. L'objet aura un id, une clé publique permettant de désigner le nouveau possesseur des fonds transférés, un montant de coins à transférer et enfin une référence à la transaction parente à laquelle cet objet est rattaché. On obtient le code suivant :

```
import java.security.PublicKey;

public class TransactionOutput {

 public String id;
 public PublicKey recipient;
 public float value;
 public String parentTransactionId;

 public TransactionOutput(PublicKey recipient, float value, String parentTransactionId) {
  this.recipient = recipient;
  this.value = value;
  this.parentTransactionId = parentTransactionId;
  this.id = Utils.applySha256(Utils.getStringFromKey(recipient)
          + Float.toString(value) + parentTransactionId);
 }

 public boolean isMine(PublicKey publicKey) {
  return (publicKey == recipient);
 }
}
```

Traitement d'une transaction

Les blocs de notre Blockchain contiendront plusieurs transactions et la Blockchain va devenir de plus en plus grande avec le temps. Pour l'anecdote, sachez que la Blockchain Bitcoin fait actuellement plus de 135 Go … Dans ces conditions, réaliser une nouvelle transaction risque de prendre un temps certain puisque nous allons devoir chercher et vérifier les transactions entrantes.
Pour éviter ce problème potentiel, nous allons stocker une liste des transactions non dépensées utilisables comme transactions entrantes directement au niveau de notre Blockchain. Cette liste sera nommée *UTXOs* afin de respecter les conventions de nommage du Bitcoin. En outre, on référence 2 instances de l'objet *Wallet* au sein de notre classe *Blockchain* afin de représenter 2 personnes souhaitant effectuer des transactions.
Nous allons maintenant pouvoir passer au gros du travail en implémentant la méthode *processTransaction*. Ajoutée au sein de la classe *Transaction*, elle sera en charge du traitement d'une nouvelle transaction sur notre Blockchain. Elle retournera un booléen indiquant si la transaction courante peut être exécutée sur la Blockchain.
La première étape va consister à vérifier la signature des données de la transaction en appelant la méthode *verifySignature* . Une fois l'intégrité des données confirmée, on associe les données des transactions entrantes de notre transaction en les récupérant depuis la liste *UTXOs* de la Blockchain. On s'assure ainsi que les fonds de ces transactions entrantes n'ont pas déjà été dépensés. L'étape suivante consiste à s'assurer que la transaction est valide. Pour cela, on fait la somme des montants des transactions entrantes au sein d'une méthode dédiée *getInputsValue*. On s'arrête là si ce montant est inférieur au montant minimum des transactions pouvant être réalisées sur la Blockchain.
Cette vérification effectuée, on génère les transactions entrantes. On crée un premier objet *TransactionOutput* représentant l'envoi réalisé au destinataire en plaçant en entrée le montant de la transaction. On crée ensuite un objet *TransactionOutput* représentant ce qu'il reste à l'expéditeur une fois l'envoi réalisé. Pour cela, on met en entrée le montant résultant de la soustraction de la somme

des montants des transactions entrantes par le montant de la transaction en cours de traitement.
L'ajout de ces transactions sortantes à la liste *UTXOs* de notre Blockchain se fait juste après. Enfin, la dernière étape est essentielle puisqu'on retire les transactions entrantes de la liste *UTXOs* des transactions non dépensées de notre Blockchain. Tout ceci donne l'ajout du code suivant à la classe *Transaction* :

```
public class Transaction {

 // ...

 public boolean processTransaction() {
  if (verifiySignature() == false) {
   System.out.println("Echec de vérification de la signature de la Transaction");
   return false;
  }

  // On associe les données des transactions entrantes
  for (TransactionInput i : inputs) {
   i.UTXO = Blockchain.UTXOs.get(i.transactionOutputId);
  }

  // On vérifie que la transaction est valide
  if (getInputsValue() < Blockchain.minimumTransaction) {
   System.out.println("Somme montant des transactions entrantes trop faible : " +getIn
putsValue());
    System.out.println("Le montant doit être plus grand que : " + Blockchain.minimum
Transaction);
    return false;
  }

  // Génération des transactions sortantes
  // Calcul montant restant
  float leftOver = getInputsValue() - value;
  transactionId = calulateHash();
  // prise en compte envoi du montant au destinataire
  outputs.add(new TransactionOutput(this.recipient, value, transactionId));
  // prise en compte montant restant pour l'expéditeur
  outputs.add(new TransactionOutput(this.sender, leftOver, transactionId));

  // Ajout aux transactions non dépensées de la Blockchain
  for (TransactionOutput o : outputs) {
   Blockchain.UTXOs.put(o.id, o);
  }

  // Suppression de ces transactions de la liste UTXOs de la Blockchain
  for (TransactionInput i : inputs) {
   if (i.UTXO == null)
    continue;
   Blockchain.UTXOs.remove(i.UTXO.id);
  }

  return true;
 }

 public float getInputsValue() {
  float total = 0;

  for (TransactionInput i : inputs) {
   if (i.UTXO == null)
    continue;

   total += i.UTXO.value;
  }

  return total;
 }

 // ...
}
```

Mise à jour du portefeuille

Maintenant qu'il nous est possible de traiter une transaction, nous allons devoir implémenter un mécanisme permettant de générer une nouvelle transaction. Cela se fait au niveau du portefeuille d'un utilisateur. Dans un portefeuille, nous devons être capables de connaître à tout moment quel montant est stocké. Pour cela, nous parcourons les transactions entrantes non dépensées se trouvant dans la liste *UTXOs* de la Blockchain et nous nous intéressons à celles appartenant au portefeuille courant. Il nous suffit ensuite d'effectuer la somme des montants de ces transactions pour connaître le montant stocké dans le portefeuille. On en profite également pour mettre à jour la liste des transactions non dépensées du portefeuille.
D'autre part, il nous reste à ajouter une méthode *sendFunds* à notre classe *Wallet* prenant en entrée l'adresse du destinataire ainsi que le montant de la transaction que nous souhaitons générer. Cette méthode se charge tout d'abord de vérifier que le portefeuille courant a assez de fonds disponibles pour envoyer un tel montant avant de créer la liste des transactions entrantes associées à la transaction en cours de création. Enfin, on crée l'objet *Transaction*, on le signe avec la clé privée du *Wallet* avant de retirer les transactions entrantes portées désormais par notre instance de *Transaction* du *Wallet*.
La mise à jour du *Wallet* nous donne le code suivant :
code complet sur le site github et ur www.programmez.com

La suite de l'article le mois prochain

Damien VAN ROBAEYS
Ingénieur Systèmes Metsys
MVP Cloud & Datacenter Management
http://www.systanddeploy.com

Créer des Dashboards dynamiques avec PowerShell

Partie 1

Dans nos métiers de l'informatique, il est souvent utile d'avoir accès à des dashboards (tableaux de bord), permettant de résumer graphiquement des actions effectuées, des états spécifiques de machines, serveurs, services… Divers outils permettent de réaliser ces Dashboards. Dans cet article, je vais vous présenter l'outil Universal Dashboard, qui permet de réaliser facilement des Dashboards graphiques (responsive) avec un design moderne, à l'aide de PowerShell.

1

Quelques exemples concrets de Dashboard viendront compléter la présentation : audit de déploiement de postes de travail, audit système, audit comptes et mots de passe AD).

Universal Dashboard késako ?

Universal Dashboard a été développé par Adam Driscoll en 2017 et édité par Ironman Software.

Avec Universal Dashboard, vous pourrez créer :

- Des Dashboards dynamiques ;
- Des formulaires ;
- Des sites Web ;
- Des API Rest.

Le design est basé sur Materials Design.css et Charts.js (1), pour la partie graphique.

Une présentation très intéressante (2) sur le produit a été réalisée par Arnaud Petitjean pour le French PowerShell User Group.

Licence Universal Dashboard ?

Universal Dashboard est un produit open source. Une version gratuite est disponible avec cependant des fonctionnalités limitées.

- Version Community (gratuit) → cette version gratuite est totalement open source. Certaines fonctionnalités telles que la possibilité d'utiliser des API Rest ne sont pas supportées.
- Version Pro (payante) → cette version offre accès à toutes les fonctionnalités, mise à jour garantie pendant 1 an, accès au support, et garantie.

Plus de détails (3).

Installer Universal Dashboard

La version de Universal Dashboard utilisée au moment de l'écriture de cet article est la 1.7.0

L'installation de Universal Dashboard s'effectue via l'installation du module depuis la PowerShell Gallery. Pour cela, utilisez la commande : **Install-module universaldashboard**

Une fois installé, vous avez accès à différentes commandes, visibles via la commande suivante : **Get-command -module universal dashboard**

Gestion des Dashboard

La gestion des Dashboards se compose en trois parties :

- Lister les Dashboards existants è **Get-UDDashboard**
- Fermer les Dashboards existants è **Get-UDDashboard | Stop-UDDashboard**
- Créer votre nouveau Dashboard

En trois étapes, réalisez la création d'un Dashboard :

1 • Création du Dashboard : **$MyDashboard = New-UDDashboard -Title "Dashboard Programmez" -Content {Contenu du dashboard}** ;

2 • Démarrage du Dashboard : **Start-UDDashboard -Port 1000 -Dashboard $MyDashboard ;**

3 • Accès au dashboard : accéder à votre Dashboard [Fig.1] via le lien suivant : http://localhost:1000 .

Le choix du port est totalement libre, vous pouvez donc choisir la valeur souhaitée.

Si vous voulez démarrer un nouveau Dashboard utilisant le même port qu'un précédent, il faudra dans ce cas fermer ce dernier.

Vous pouvez changer simplement le titre en utilisant l'attribut « -Title », ou encore la couleur du header et footer en utilisant l'attribut « -Color ». [Fig. 1]

Insérez les composants graphiques et autres dans le contenu du Dashboard.

Composants graphiques Universal Dashboard

Comme mentionné précédemment, Universal Dashboard comporte différents composants (graphiques, pages…).

(1) https://www.chartjs.org/
(2) https://youtu.be/9BCoX3WJYf4
(3) https://poshtools.com/2018/03/31/changes-universal-dashboard-licensing/
(4) https://github.com/damienvanrobaeys/Universal-Dashboard-Programmez

Une liste complète et détaillée des commandes est disponible ici (4).
Parmi ceux-ci, on retrouve les modèles ci-dessous (avec la commande associée pour créer le composant) :

- Camembert, Histogramme, Donuts… → New-UDChart
- Grille (Grids) → New-Grid
- Moniteur (Monitor) → New UDMonitor
- Tableau (Table) → New-UDTable

Dans l'exemple suivant [Fig. 8], nous allons voir un Dashboard permettant de monitorer les installations des postes de travail d'un parc informatique.
Ce monitoring s'effectue avec Microsoft Deployment Toolkit (MDT) et traduit via Universal Dashboard.
Les scripts complets de cet article sont disponibles ici (4).

Gestion de l'interface (Layout)

Plusieurs commandes permettent de gérer l'affichage principal. C'est un point important pour comprendre comment disposer ses composants.

- New-UDColumn → Création d'une colonne
- New-UDRow → Création d'une ligne

La disposition de l'affichage se compose avec un maximum de 12 colonnes et autant de lignes que l'on souhaite. Lors de la création d'un Dashboard, il faut donc imaginer son écran divisé en 12 colonnes. L'exemple suivant [Fig.2], montre comment s'organise l'affichage en fonction de la taille des colonnes.
Pour afficher un composant sur l'ensemble de la largeur, il faut lui attribuer la taille de 12, comme sur la 1ere ligne de l'exemple.
La commande New-UDRow, nous permettra d'ajouter des lignes.
Le code de l'exemple [Fig. 2], est le suivant :

```
# Première ligne
New-UDRow -Columns {
    New-UDColumn -Size 12 {
        New-UDCard -Title "Taille 12" -FontColor "White" -BackgroundColor "#5290E9"}
    }

# Seconde ligne
New-UDRow -Columns {
    New-UDColumn -Size 6 {
        New-UDCard -Title "Taille 6" -FontColor "White" -BackgroundColor "#5290E9"}

    New-UDColumn -Size 6 {
        New-UDCard -Title "Taille 6" -FontColor "White" -BackgroundColor "#71B37C"}
    }

# Troisième ligne
New-UDRow -Columns {
    New-UDColumn -Size 2 {
        New-UDCard -Title "Taille 2" -FontColor "White" -BackgroundColor "#5290E9"}

    New-UDColumn -Size 4 {
        New-UDCard -Title "Taille 4" -FontColor "White"  -BackgroundColor "#71B37C"}

    New-UDColumn -Size 6 {
        New-UDCard -Title "Taille 6" -FontColor "White"  -BackgroundColor "#2b5797"}
    }
}
```

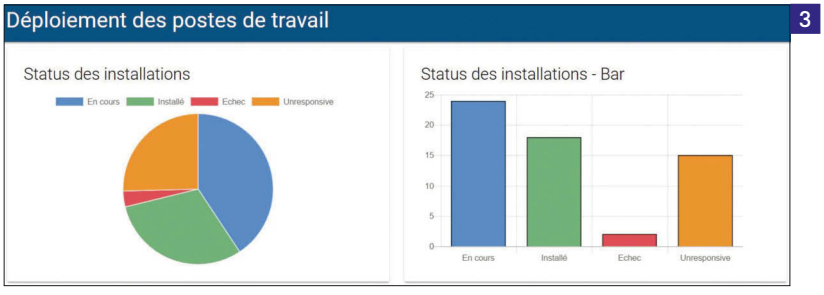
Il est donc possible, assez simplement, d'organiser ses Dashboards. Pour la plupart des composants graphiques, il est possible de spécifier le titre, la couleur de police ou de fond…
Un paramètre très utile, « -RefreshInterval », permet de spécifier à quelle fréquence vous souhaitez rafraîchir les données du Dashboard. Le paramètre – RefreshInterval permettra de spécifier à quelle fréquence les données d'un graphique seront actualisées. Par exemple, RefreshInterval 5 permettra de rafraîchir des données toutes les 5 secondes.

Création d'un graphique (chart)

Plusieurs types de graphiques disponibles : barre (Bar), ligne (Line), donuts (Doughnuts), camembert (Pie).
Ci-dessous un exemple de chart utilisant le modèle Doughnuts et Bar [Fig. 3].
Le code utilisé pour afficher ces composants est le suivant :

```
New-UDRow -Columns {
    New-UDColumn -Size 5 {
    New-UDChart -Title "Status des installations" -Type "Doughnut" -Endpoint {
        $Data | Out-UDChartData -LabelProperty "Install"  -Dataset @(
            New-UDChartDataset -Label "Install" -DataProperty Count -BackgroundColor @
("#5290E9","#71B37C","#E14D57","#EC932F") -BorderColor @("white", "white", "white","white")
 -BorderWidth 1

        )
    } -Options @{cutoutPercentage = 0}

}
    New-UDColumn -Size 5 {
    New-UDChart -Title "Status des installations" -Type "Bar" -Endpoint {
        $Data | Out-UDChartData -LabelProperty "Install"  -Dataset @(
            New-UDChartDataset -Label "Install" -DataProperty Count -BackgroundColor @("#5290E9",
"#71B37C","#E14D57","#EC932F") -BorderColor @("white", "white", "white","white") -BorderWidth 1

        )
    } -Options @{legend  = $false}
```

2



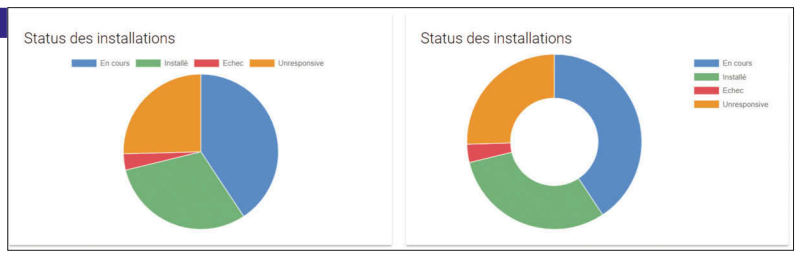
3

```
}
}
```

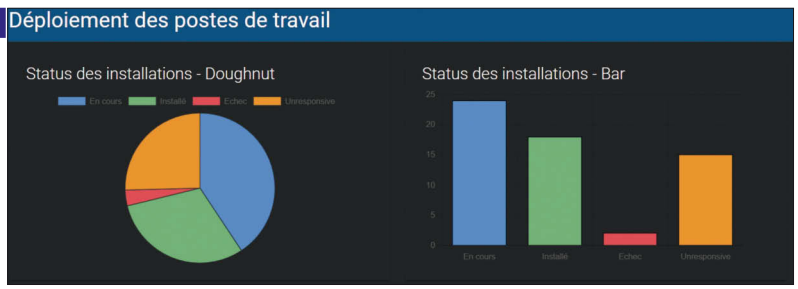
Deux aspects de Doughnuts sont disponibles [Fig. 4]
Le code pour cet exemple est le suivant :

```
New-UDRow -Columns {
    New-UDColumn -Size 5 {
    New-UDChart -Title "Status des installations" -Type "Doughnut" -Endpoint {
        $Data | Out-UDChartData -LabelProperty "Install"  -Dataset @(
            New-UDChartDataset -Label "Install" -DataProperty Count -BackgroundColor @("#5290E9",
"#71B37C","#E14D57","#EC932F") -BorderColor @("white", "white", "white","white") -Border
Width 1

        )
    } -Options @{cutoutPercentage = 0}

}
    New-UDColumn -Size 5 {
    New-UDChart -Title "Status des installations" -Type "Doughnut" -Endpoint {
        $Data | Out-UDChartData -LabelProperty "Install"  -Dataset @(
            New-UDChartDataset -Label "Install" -DataProperty Count -BackgroundColor @("#5290E9",
"#71B37C","#E14D57","#EC932F") -BorderColor @("white", "white", "white","white") -Border
Width 1
```

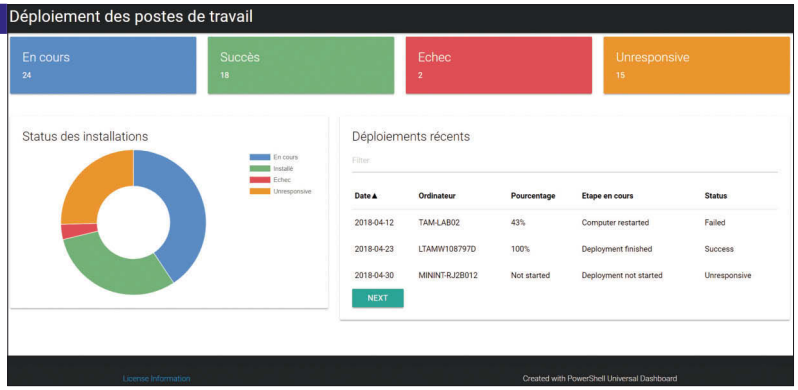
4



6 Déploiement des postes de travail



8 Déploiement des postes de travail

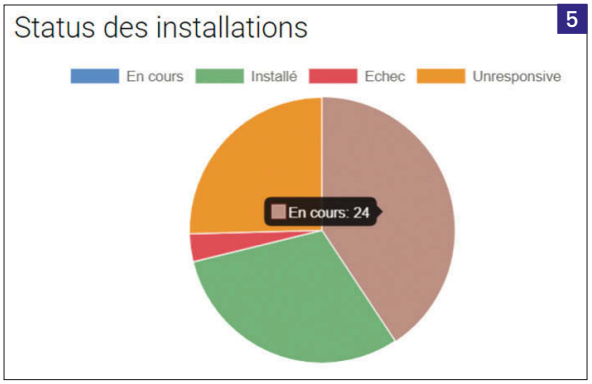



```
        )
    } -Options $Options

}
}
```

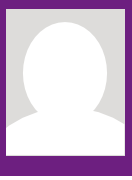
Lorsque vous passez le curseur de la souris sur un graphique, la valeur associée s'affiche [Fig. 5].
Autre mise en forme possible, changer la couleur du fond pour donner un aspect plus sombre [Fig. 6].
Pour cela, utilisez le code ci-dessous :

```
$Colors = @{
    BackgroundColor  = "#FF252525"
    FontColor        = "#FFFFFFFF"
}

$Dashboard = New-UDDashboard -Title "Déploiement des postes de travail" @Colors -Content {
    New-UDRow -Columns {
        New-UDColumn -Size 5 {
        New-UDChart -Title "Status des installations" -Type "Doughnut" @Colors -Endpoint {
            $Data | Out-UDChartData -LabelProperty "Install"  -Dataset @(
                New-UDChartDataset -Label "Install" -DataProperty Count -BackgroundColor
@("#5290E9","#71B37C","#E14D57","#EC932F") -BorderColor @("white", "white", "white",
"white") -BorderWidth 1

            )
        } -Options @{cutoutPercentage = 0}

    }
        New-UDColumn -Size 5 {
        New-UDChart -Title "Status des installations" -Type "Bar" @Colors -Endpoint {
            $Data | Out-UDChartData -LabelProperty "Install"  -Dataset @(
                New-UDChartDataset -Label "Install" -DataProperty Count -BackgroundColor
@("#5290E9","#71B37C","#E14D57","#EC932F") -BorderColor @("white", "white", "white",
"white") -BorderWidth 1

            )
        } -Options @{legend  = $false}

    }
    }
}
```

5



La suite de l'article le mois prochain



Duy Anh Pham
développeur Clevermind

niveau 200

Principes avancés de conception objet

Partie 2

La POO, ou Programmation Orientée Objet, permet de représenter un concept, une idée ou une entité du monde physique par des entités appelées objets ayant des propriétés intrinsèques et exposant des opérations publiques pour les manipuler. Ainsi les objets peuvent être vus comme des briques rendant des services aux autres objets et donc réutilisables. L'interaction entre les objets via leurs relations permet de concevoir et réaliser les fonctions attendues.

Principe de substitution de Liskov

L'OCP montre l'importance de l'héritage en séparant la partie commune et variable qui est extensible. De manière générale, l'abstraction avec l'utilisation des interfaces permet de découpler les classes en faisant reposer une classe sur une abstraction de classes. 1

La technique présentée ci-dessus joue un rôle primordial dans la modularité des applications et n'est efficace que si l'abstraction est parfaitement identifiable. L'interface I doit fournir une bonne abstraction des classes Ci et ces dernières doivent s'y conformer. C'est précisément le sens du principe de substitution de Liskov.

C'est Barbara Liskov qui a donné la définition du sous typage : **un sous type S doit être substituable à son type de base T dans toute l'application où T est utilisé sans causer de comportement non désiré dans le programme.**

Exemple 1

Une équipe développe une librairie de fichiers XML destinée aux projets Java de toute l'entreprise en exposant une classe D avec les opérations de lecture et écriture : read() ; write().

Les projets ont le choix de l'implémentation parmi DOM (D) et SAX (S) proposées par la librairie comme sous classes. 2

La classe s'appuyant sur SAX hérite de la classe de base pour réutiliser du code de journalisation des évènements lors de la lecture et écriture, ce qui est tout à fait louable puisque la duplication de code est éliminée. Une équipe choisit DOM pour la facilité de l'emploi des objets et poursuit le développement de son application jusqu'au jour où un test de charge provoque une erreur de mémoire de type OutOfMemory Error à la lecture d'un fichier XML conséquent. Une demande de support à l'équipe technique leur permet de résoudre leur problème en changeant DOM par SAX qui est peu consommateur en mémoire.

L'ennui à présent est que l'écriture repasse en mode DOM puisque SAX est une API de parsing uniquement. Le projet était dans l'impossibilité de générer des fichiers XML volumineux pour les envoyer à leurs différents partenaires. S n'est pas substituable à D.

Exemple 2

Un éditeur souhaite créer un jeu vidéo simulant l'activité humaine. Il sous-traite le développement de l'homme à une équipe tierce pour se focaliser sur l'infrastructure d'une ville et les activités urbaines. Une classe de base Human leur est fournie sous forme de librairie avec plusieurs sous-classes comme Man, Woman, et Baby. 3

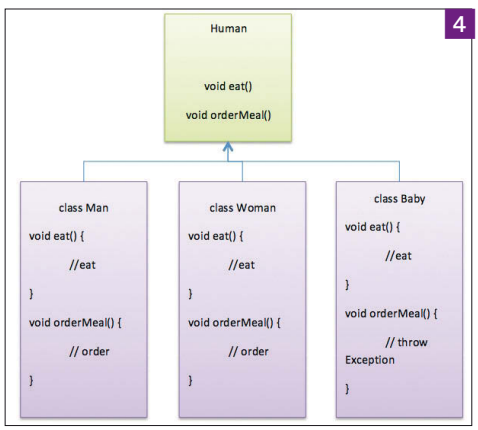
L'interface Human propose des méthodes comme eat() et orderMeal(). 4

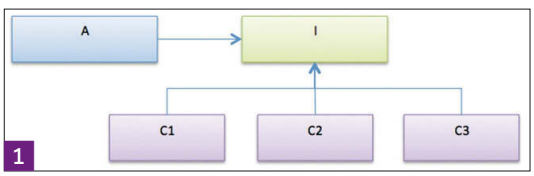
L'éditeur intègre dans la librairie et l'utilise pour simuler l'activité humaine dans la ville. Un test est réalisé en chargeant la partie dans un restaurant pour qu'un humain se nourrisse. Le test est concluant pour un adulte comme Man ou Woman, mais pour un Baby le test se solde par une erreur. En effet, un bébé n'est pas autonome pour être capable de commander un plat. La méthode orderMeal() renvoyait une exception UnsupportedOperation Exception. Baby n'est pas substituable à Human dans ce cas d'utilisation. Ces exemples montrent que pour que la substitution soit efficace, il est important que les opérations de toutes les sous-classes respectent bien un contrat établi.

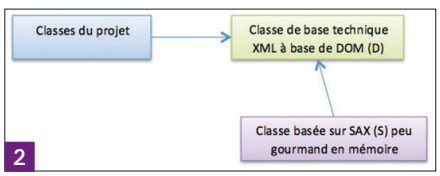
Conception par contrat

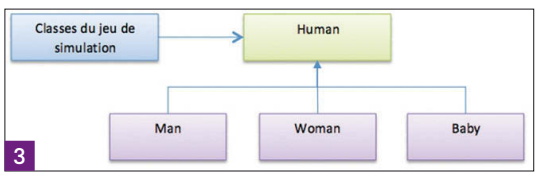
La définition met en exergue l'importance du rôle de la classe de base qui se présente comme une offre de service fournie par chaque sous classe. En langage objet, cela signifie que la classe de base A est une interface exposée que les implémentations doivent respecter. Ce concept rejoint celui du "Design by contract" de Bertrand Meyer, l'interface représentant un véritable contrat passé entre chaque sous-classe et les classes susceptibles de l'utiliser.

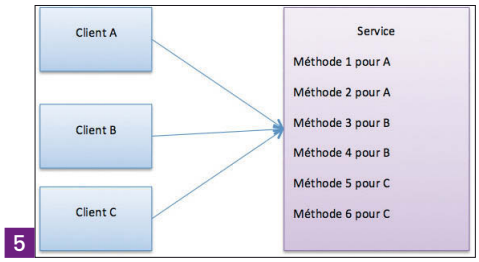
En s'appuyant sur le mécanisme d'héritage et plus particulièrement le polymorphisme, le principe de substitution s'oppose à une pratique très répandue dans laquelle l'héritage permet de factoriser du code dans la classe de base pour être réutilisé par plusieurs sous-classes. Du point de vue langage, ceci est tout à fait « légal » puisque le code compile. Il est cependant



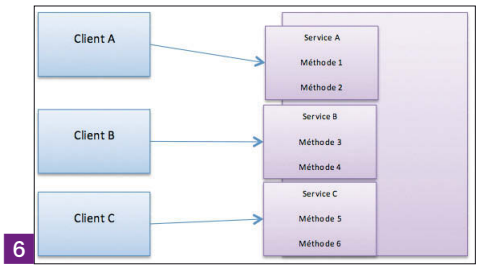




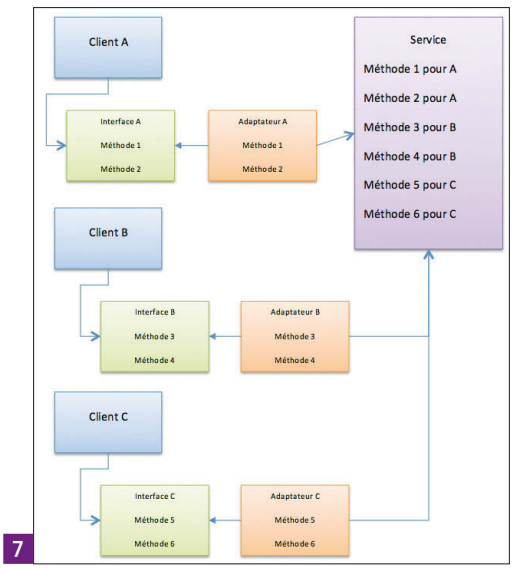




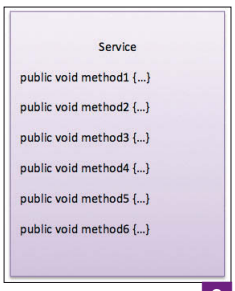
5



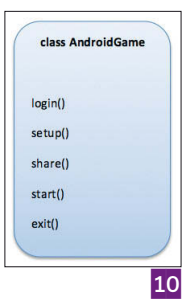
6



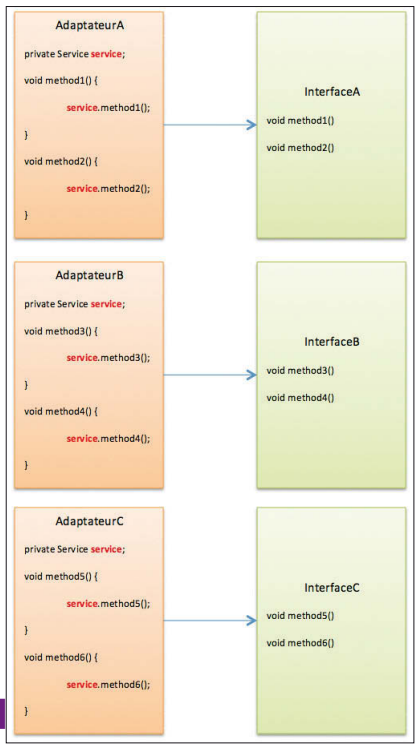
7



9



10



8

plus efficace d'utiliser la composition, en externalisant dans un module dédié à cette responsabilité (voir SRP).

Jusqu'où peut aller la substitution ?

Selon le principe, la substitution est parfaite tant que le contrat est respecté. A mesure que l'application évolue, tôt ou tard, une classe finira par ne pas respecter le contrat établi par l'interface et 2 choix s'offrent alors au développeur :

- l'interface reste fermée et impose donc de recourir au downcast (utilisation de instanceof en Java avec cast) entraînant une violation de l'OCP ;
- l'interface est étendue pour couvrir ce cas particulier et impose donc aux autres classes qui l'implémentent une partie qui ne les concernent pas. Ce sont typiquement des méthodes sans corps ou lançant une exception de type UnsupportedOperationException. Ceci entraîne cette fois une violation du LSP.

La 1ère solution est de loin préférable pour respecter l'intégrité du contrat. En d'autres termes, un downcast isolé pour implémenter un cas particulier est préférable plutôt que de corrompre l'interface et surtout masquer le problème. Dans le cas du jeu de simulation évoqué plus haut, un test avec instanceof Baby doit être fait pour éviter un plantage dans la partie de jeu au restaurant. Sur des projets de grande envergure, il n'est pas rare de voir de nombreuses classes implémentant des méthodes héritées sans corps, ce qui rend le refactoring d'autant plus difficile car le développeur n'a pas forcément connaissance de la méthode d'invocation sur ces méthodes (réflexion, composant distribué …).

Principe de ségrégation des interfaces

Le principe stipule que le client ne doit voir que les services dont il a besoin.

Autrement dit la dépendance d'une classe vers une autre doit être restreinte à l'interface la plus petite possible.

Lorsque le principe de responsabilité unique est respecté avec un rôle bien défini et une forte cohésion, elle peut être utilisée par différents clients sans que ces derniers n'utilisent des fonctions communes. 5

Quand appliquer :

L'inconvénient est que tous les clients voient les opérations exposées par le service :

- le client n'utilise qu'une partie de l'interface qui l'intéresse
- chaque client peut être impacté par les changements d'une interface qu'il n'utilise pas

Pour y remédier, il faut donc séparer l'interface en autant d'interfaces pour chaque client 6

Comment appliquer :

Utiliser le pattern adaptateur permet d'exposer uniquement les opérations d'un service pour le client et encapsule la classe concrète en délégant l'appel. 7

Chacun des clients utilise l'interface dont il a vraiment besoin et l'implémentation associée utilise le même service sous-jacent pour effectuer le traitement demandé. Code des interfaces et adaptateurs respectifs 8 9

Principe d'inversion des dépendances

Les modules de haut niveau ne doivent pas dépendre de modules de bas niveau. Tous deux doivent dépendre d'abstractions. Les abstractions ne doivent pas dépendre de détails. Les détails doivent dépendre d'abstractions.

Problèmes des architectures monolihiques

Par exemple : une classe AndroidGame peut avec les opérations suivantes : 10

Lorsque nous jouons à un jeu sur un smartphone Android (ceci est vrai également pour un jeu sur iPhone), celui-ci propose une identification pour sauvegarder la partie et partager les exploits de deux manières :

- avec un identifiant de l'éditeur du logiciel. L'utilisation d'un module de connexion aux bases de données est nécessaire pour effectuer les opérations transactionnelles ;
- avec un compte Facebook, avec un module de communication vers une application extérieure de type Web service. 11

Il est souhaité que les modules métiers soient les plus réutilisables possibles. Or ces derniers sont construits sur d'autres modules de bas niveau et cela ceci pose deux problèmes :

- Les modules de haut niveau sont impactés lorsqu'un module de bas niveau est modifié ;
- Il n'est pas possible de réutiliser les modules de haut niveau indépendamment

de ceux de bas niveau. En d'autres termes, il n'est pas possible de réutiliser la logique d'une application en dehors du contexte technique dans lequel elle a été développée.

Comment appliquer :

L'inversion des dépendances par emploi de l'abstraction. Selon le principe, la relation des dépendances doit être inversée : les modules de bas niveau doivent dépendre d'une abstraction qui sera utilisée par les modules de haut niveau. 12

De manière générale : quand une situation se présente comme suit 13

L'inversion consiste à introduire une interface I dont A dépendra directement et dont B dépend également et qu'il doit implémenter. 14

Nous reconnaissons le mécanisme de délégation abstraite employé dans l'OCP. L'ouverture/fermeture est obtenue en inversant les dépendances de sorte que A ne soit plus impacté par les changements dans B.

Les classes A et B dépendent de I pour compiler. Cependant A a besoin de B au runtime, c'est à dire à l'exécution, pour fonctionner et remplir pleinement ses fonctions. Par exemple, dans le monde JAVA/JEE, l'API Servlet d'un projet Web est nécessaire à la compilation pour créer des Servlets, mais au runtime l'implémentation de l'API Servlet est fournie par le serveur d'applications ou conteneur de Servlet sur lequel l'application est déployée.

Vers des frameworks métier

Ce principe conduit à des applications dont la logique métier est réutilisable quel que soit le contexte technique. En effet, la partie métier forme un « framework » qui permet de développer une même application dans plusieurs contextes techniques différents.

Organisation de l'application en modules

Principe de réutilisabilité de l'équivalence de livraison

La granularité en termes de réutilisation est le package. Seuls des packages livrés sont susceptibles d'être réutilisés.

Reuse-Release Equivalence Principle - REP : signifie que pour réutiliser du code, il doit être livré complet dans une boîte noire. Les utilisateurs de ce code doivent être protégés des changements car ils doivent être libres de décider quand intégrer le package dans leur code. Cependant la réutilisabilité n'est efficace que si :

- Le code reste la propriété de son auteur qui a la charge de le maintenir et faire évoluer ;
- Le code est réutilisé tel quel avec l'API publique.

En effet il est très néfaste de vouloir intégrer le code tierce dans son propre code et de le patcher pour ses besoins avec des effets de bord que seul son auteur maîtrise.

L'ensemble des classes et interfaces de l'API doivent être livrées et versionnées. La granularité adéquate est le package car c'est le niveau approprié pour livrer cet ensemble. Par exemple migrer la librairie commons-lang d'Apache, permettant de manipuler des classes de base Java, de la version 2.4 vers 2.6 peut se faire sans aucun risque.

Principe de réutilisabilité commune

Réutiliser une classe d'un package, c'est réutiliser le package entier.

Common Reuse Principle - CRP

Il est rare d'utiliser une classe seule surtout si le SRP est respecté. Les packages doivent être constitués de classes susceptibles d'être réutilisées ensemble. Dans ce cas, il est plus efficace de les intégrer dans un même package, ce qui facilite l'utilisation de librairie par l'utilisateur.

A l'inverse en termes de dépendances, utiliser une classe d'une librairie revient à utiliser toute la librairie. Il faut veiller à ne pas inclure 2 classes totalement indépendantes dans un même package car l'utilisateur est forcé de dépendre d'une autre classe B dont il n'a pas besoin alors qu'il utilise une classe A.

Principe de fermeture commune

Les classes impactées par les mêmes changements doivent être placées dans un même package.

Common Closure Principle - CCP

Le principe stipule qu'il faut regrouper dans un même package les classes impactées par un même changement.

Gestion de la stabilité de l'application

Principe des dépendances acycliques

Les dépendances entre packages doivent former un graphe acyclique.

Acyclic Dependencies Principle – ADP

L'objectif de la décomposition en package est de limiter la propagation des impacts lorsqu'un package est modifié. Les changements intervenus sur une classe impactent immédiatement les autres classes du même package mais n'impactent les autres packages qui en dépendent que si une nouvelle version du package est livrée. 15

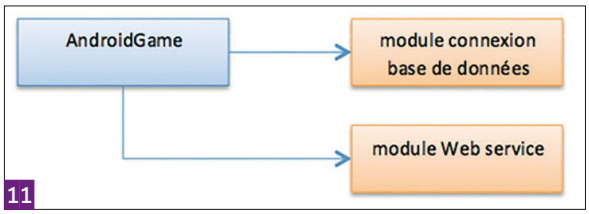
Les modifications d'une classe dans le package A ne sont propagées qu'aux packages B et C que lorsque ceux-ci décident d'utiliser une nouvelle version de A, et aux packages D et E que lorsque ceux-ci décident d'utiliser une nouvelle version de C.

La propagation des changements est guidée par les dépendances entre packages et forment un graphe orienté. L'organisation de ces dépendances forme un élément fondamental dans l'architecture de l'application. Le principe des dépendances acycliques stipule que les dépendances entre packages doivent former un graphe acyclique orienté. Que se passerait-il si le graphe devient cyclique ? 16

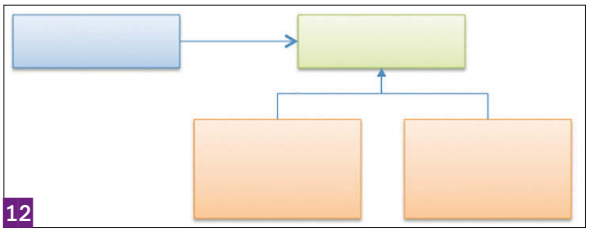
Cette fois le package A dépend du package E.

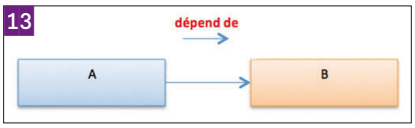
- A dépend de E pour compiler ;
- E dépend de C pour compiler ;
- C a besoin de A pour compiler.

Si A est modifié, alors il peut devenir impossible de recompiler ne serait-ce qu'un seul package car C a besoin de l'ancienne

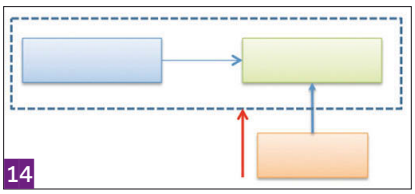


11


12



13


14

version de A pour compiler. C et E doivent être recompilés voire modifiés avant de compiler A. Les packages doivent donc évoluer ensemble. Si un graphe cyclique apparaît dans l'application, c'est peut être le signe que l'ensemble des classes doivent être reconsidérées pour être réunies et livrées dans un même package. 17

Une autre solution consisterait à utiliser l'inversion des dépendances pour briser le cycle 18

Le package A dépend de l'interface E puisqu'il dépendait initialement directement du package E. Ce dernier implémente l'interface E. L'inversion permet de changer le sens d'une dépendance et d'éliminer un cycle. JDepend est un outil d'analyse des dépendances des packages Java. Le programme existe sous forme de plugin Eclipse pour faciliter l'intégration lors du développement de code.

Principe de relation dépendance/stabilité

Un package doit dépendre uniquement de packages plus stables que lui.

Stable Dependencies Principle - SDP

La stabilité ou instabilité est liée au couplage entre 2 modules. Elle mesure le degré de fragilité du module si des changements s'opèrent dans les dépendances extérieures. JDepend permet de calculer cette métrique également et peut être intégré dans le processus de développement pour améliorer le code.

L'instabilité correspond à un ratio entre les couplages efférents (Ce) et afférents (Ca) de telle sorte que $I = \frac{C_e}{C_{e+}C_a}$. Cette métrique est un indicateur de stabilité par rapport à la mise à jour d'autres packages.

Plus formellement, pour un module donné :

- Plus le nombre de modules dont il dépend est grand (couplage efférent), plus il est susceptible d'être impacté par des modifications d'un de ces modules, et donc moins il est stable. 19
- Plus le nombre de modules dépendant de ce module est grand (couplage afférent), plus les modifications de ce module sont coûteuses, et donc plus il est stable. Cela traduit la responsabilité du module dans l'ensemble du code 20

Selon cette définition, la stabilité du module est :

- Maximale si le module n'utilise aucun autre module et se trouve lui-même utilisé par un grand nombre de modules (couplage efférent faible et couplage afférent fort) ;
- Minimale si le module utilise de nombreux autres modules alors qu'il n'est utilisé lui-même par aucun autre module (couplage efférent fort et couplage afférent faible).

La mesure de l'instabilité est calculée comme suit :

$I = \frac{C_e}{C_{e+}C_a}$ avec I variant entre 0 et 1

Dans les configurations suivantes :

- Stabilité maximale : Ce = 1 et Ca = infini donc I tend vers 0 ;
- Stabilité minimale : Ce = infini et Ca = 1 donc I tend vers 1.

Par exemple dépendre directement du JDK pour manipuler l'API Collection est maximale par rapport à utiliser une librairie tierce. De base tout projet Java dépend du JDK pour compiler.

Le principe de relation dépendance/stabilité stipule qu'un module doit dépendre uniquement de modules plus stables que lui. En effet, l'impact sur les changements dans ces derniers serait amorti, maximisant la stabilité globale de l'application.

Principe de stabilité des abstractions

Les packages les plus stables doivent être les plus abstraits.
Les packages instables doivent être concrets.
Le degré d'abstraction d'un package doit correspondre à son degré de stabilité.

Stable Abstractions Principle - SAP

Ce principe stipule que les interfaces et les classes qui l'implémentent doivent être dans des packages différents.

En effet, les interfaces ne contiennent que la signature des méthodes publiques (les méthodes par défaut existent depuis Java 8) avec éventuellement des constantes et sont mises à disposition pour d'autres équipes de développement. Si des bugs existent, alors leurs corrections ne concernent que les implémentations et sont ainsi isolées dans un package dédié.

Conclusion

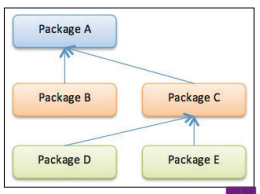
La conception doit placer le contrôle des dépendances au cœur de son activité pour limiter les impacts des changements. Le coût des modifications engendrées serait alors réduit et les objectifs recherchés d'extensibilité, robustesse et réutilisabilité seraient atteints. Plusieurs principes montrent le rôle majeur des interfaces en termes d'extensibilité et robustesse :

- Elles servent de pare-feu arrêtant la propagation des changements d'un module sur les modules afférents (inversion de dépendance, abstraction et stabilité).
- L'héritage doit être davantage considéré comme une implémentation de l'interface plutôt qu'un moyen de factoriser du code. L'interface représente un contrat à respecter pour le code client et les classes qui l'implémentent rendant la substitution opérationnelle grâce au polymorphisme (OCP et principe de substitution de Liskov).
- Avec le principe de ségrégation, une classe peut implémenter plusieurs interfaces répondant à plusieurs services et par conséquent le client n'utiliserait qu'une seule interface avec uniquement les méthodes nécessaires.

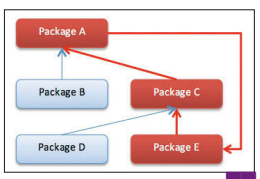
La robustesse s'obtient aussi en regroupant les classes fonctionnant ensemble :

- Pour isoler au même endroit les changements induits par les classes efférentes (CCP) ;
- Pour éviter les dépendances cycliques (ADP) ;
- En séparant les interfaces des classes concrètes dans 2 packages différents pour (SDP et SAP) la stabilité de l'application est augmentée.

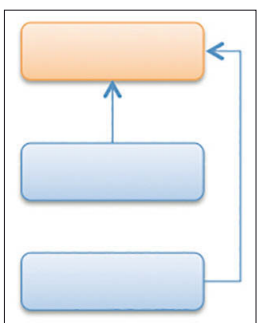
Ces principes constituent un cadre solide pour concevoir une application extensible et robuste. •

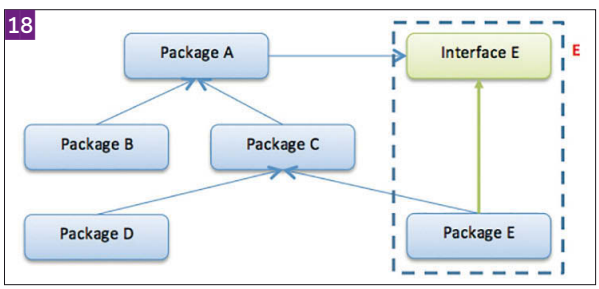


15

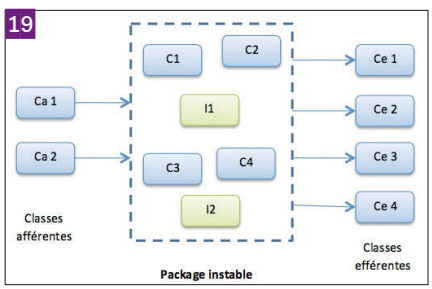


16

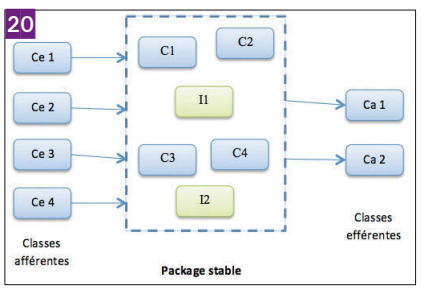

17



18



19



20

Charles DE VANDIERE
Développeur Infinite Square

INFINITE SQUARE

GraphQL : exposez vos données dynamiquement dans une API REST

niveau 100

Partie 2

Il est parfois nécessaire de développer une API permettant de retourner uniquement les champs demandés par l'appelant de façon complètement dynamique.

IMPLÉMENTATION DANS ASP.NET CORE

Une fois cette présentation théorique (voir n°221) faite nous allons enfin mettre la main dans le code et je vais vous montrer comment implémenter GraphQL sur une API ASP.NET Core en utilisant le projet open-source GraphQL for .NET (https://github.com/graphql-dotnet/graphql-dotnet).

Exposer les APIs de requêtage

Définition des types exposés

Dans un premier temps, il nous faut décrire les types qu'expose notre API. On va donc avoir d'un côté nos objets fonctionnels traditionnels écrits sous la forme de POCO (Plain Old C# Objects), et de l'autre côté des entités GraphQL. Ces entités GraphQL doivent toutes, en utilisant GraphQL for dotnet, dériver de la classe de base ObjectGraphType générique prenant comme paramètre de type le POCO fonctionnel exposé.

Propriétés simples

Prenons l'exemple d'un type Droid définissant deux propriétés simples Id et Name de cette manière :

```
public class Droid
{
  public string Id { get; set; }
  public string Name { get; set; }
}
```

Il nous faut alors créer un type « GraphQL » pour le définir. On l'appelle donc DroidType et on le fait dériver de l'ObjectGraphType <Droid> de cette manière :

```
public class DroidType : ObjectGraphType<Droid>
{
  public DroidType()
  {
    Name = "Droid";
    Description = "A mechanical creature in the Star Wars universe.";
    Field(d => d.Id).Description("The id of the droid.");
    Field(d => d.Name, nullable: true).Description("The name of the droid.");
  }
}
```

La propriété Name est présente sur le type de base et permet de donner un nom GraphQL au type qui n'est pas déduit automatiquement de votre type POCO. Ce nom sera aussi visible dans l'onglet Documentation Explorer de GraphiQL.

La propriété Description permet de donner… (Suspens intenable et roulement de tambour)... une description au type ou au champ qui sera également visible dans la Documentation Explorer de GraphiQL. La description n'est pas obligatoire si le champ est auto-descriptif mais cela reste une bonne pratique de la renseigner.

Propriétés de « navigation »

Partons maintenant sur le cas des propriétés plus complexes que sont les listes d'entités liées. Imaginons par exemple que notre classe Droid expose une liste d'épisodes où le Droïde apparaît sous la forme d'une propriété AppearsIn que nous ajoutons à notre POCO :

```
public class Droid
{
  public string Id { get; set; }
  public string Name { get; set; }
  public List<Episode> AppearsIn { get; set; }
}

public class Episode
{
  public string Name { get; set; }
  public int Year { get; set; }
  public int Order { get; set; }
}
```

La propriété AppearsIn étant complexe, nous ne pouvons pas l'ajouter aussi simplement que précédemment en tant que field dans l'ObjectGraphType.

Il faut pour cela utiliser une surcharge de la méthode « Field » prévue pour les types plus complexes. Cette méthode générique prend le type GraphQL retourné en « paramètre de type » (dans notre cas : ListGraphType<EpisodeType> avec ListGraphType étant un type GaphQL représentant une collection d'objets). Les paramètres d'appel à fournir seront alors :

- Le nom à exposer ;
- La fonction étant en mesure de retourner la valeur de la propriété complexe. Cette fonction prend comme paramètre le contexte de résolution contenant l'objet source (ici un Droid).

```
public class DroidType : ObjectGraphType<Droid>
{
  public DroidType()
  {
    Name = "Droid";
    Description = "A mechanical creature in the Star Wars universe.";
```

```
    Field(d => d.Id).Description("The id of the droid.");
    Field(d => d.Name, nullable: true).Description("The name of the droid.");
    Field<ListGraphType<EpisodeType>>(
      "appearsIn",
      resolve: context => context.Source.AppearsIn);
  }
}
```

EpisodeType est l'ObjectGraphType du POCO Episode que je n'ai pas détaillé ici mais que vous pouvez facilement imaginer à partir des exemples précédents. À noter qu'il existe également FieldAsync qui permet de faire des traitements asynchrones dans la méthode permettant de fournir la valeur de la propriété complexe.

Définition des Queries

Maintenant que nous avons défini les types GraphQL, nous allons écrire en .NET les queries permettant de les obtenir. Il s'agit finalement d'un objet GraphQL classique (dérivant de ObjectGraphType donc) qui définit les différentes queries sous la forme de fields. Ces fields définissent les paramètres possibles et attendus pour la query sous la forme d'un objet QueryArguments et fournissent la méthode resolve permettant de récupérer les éléments.
Voici par exemple la définition de la Query « droid » permettant d'obtenir un droid en fournissant obligatoirement (cela est imposé par l'utilisation de NonNullGraphType) son identifiant.

```
public class StarWarsQuery : ObjectGraphType<object>
{
  public StarWarsQuery(StarWarsData data)
  {
    Name = "Query";
    Field<DroidType>(
      "droid",
      arguments: new QueryArguments(
        new QueryArgument<NonNullGraphType<StringGraphType>>
        {
          Name = "id",
          Description = "id of the droid"
        }
      ),
      resolve: data.GetDroidById(id);
    );
  }
}
```

S'il fallait définir une nouvelle query, on aurait défini un champ Field de plus sur ce même objet… Une query n'est vraiment rien d'autre qu'un champ sur un objet GraphQL.
Ici, l'objet StarWarsData représente notre couche d'accès aux données. Cela pourrait être un contexte EntityFramework, une collection d'objets en mémoire, ou tout autre système d'accès aux données. Dans notre exemple, « data.GetDroidById(id) » retourne un Droid de type POCO (il n'hérite pas de ObjectGraphType). Toute la magie est opérée ensuite par GraphQL pour retourner l'objet tel qu'il a été décrit dans la classe DroidType.
Si votre fournisseur de données est asynchrone, vous pouvez utiliser FieldAsync. A ce moment-là, la méthode fournie dans le resolve pourra être asynchrone.

Définition du schéma

Maintenant que les types et les queries sont prêts, nous allons devoir définir le schéma GraphQL.
Le schéma GraphQL représente toutes les opérations pouvant être exécutées sur votre API.

```
public class StarWarsSchema : Schema
{
  public StarWarsSchema(IDependencyResolver resolver)
    : base(resolver)
  {
    Query = resolver.Resolve<StarWarsQuery>();
  }
}
```

Injection de dépendances

Maintenant que nous avons développé toutes les classes nécessaires à GraphQL il ne faut pas oublier de les ajouter à l'injecteur de dépendances de notre site Asp.Net CORE, sinon GraphQL ne pourra pas fonctionner. Les différents types GraphQL sont ajoutés de manière transiente (une nouvelle instance à chaque requête http) et le schéma est ajouté sous la forme d'un singleton.

```
service.AddTransient<StarWarsData>();
service.AddTransient<DroidType>();
service.AddTransient<EpisodeType>();
service.AddTransient<StarWarsQuery>();
service.AddSingleton(new StarWarsSchema(new FuncDependencyResolver(type =>
 service.GetRequiredType(type))));
```

Requêtage via un service

Maintenant que tout est prêt, il ne nous reste plus qu'à appeler la query donnée sous la forme d'une chaîne de caractères. Pour cela nous créons et utilisons un service Asp.Net Core nommé GraphQLService dans lequel nous injectons le schéma GraphQL. Ce service définit une méthode ExecuteQueryAsync prenant la query (un string donc) et retourne un résultat d'exécution sous la forme d'un objet GraphQl.Net : ExecutionResult.
Ce résultat est produit très simplement via l'instanciation d'un objet DocumentExecuter permettant de jouer une requête sur un schéma de cette manière :

```
public class GraphQLService : IGraphQLService
{
  private readonly ISchema _query;
  public GraphQLService(ISchema query)
  {
    _query = query;
  }
  public async Task<ExecutionResult> ExecuteQueryAsync(string query)
    => await new DocumentExecuter()
      .ExecuteAsync(options =>
      {
        options.Schema = _query;
        options.Query = query;
      });
}
```

Ce service devra être appelé depuis l'endpoint d'un controller asp.net tel que celui-ci :

```
public async Task<IActionResult> GetAsync([FromQuery] string query)
{
  ExecutionResult result = await _graphQLService.ExecuteQueryAsync(query);

  return Ok(result);
}
```

On peut alors le tester avec cette requête d'exemple :

```
{
  droid(id: "AF669ABD-AAEB-4C7B-870D-0360FDFA02D5") {
    id
    name
    appearsIn {
      name
      year
      order
    }
  }
}
```

Le résultat que nous retourne l'API est alors le suivant : il s'agit de l'ExecutionResult converti en JSON.

Voir code complet sur www.programmez.com

Les Mutations

Pour rappel, la mutation est une action GraphQL permettant de créer, modifier ou supprimer un objet.

Input Type – définition des paramètres

En première partie de cet article, nous avons créé des ObjectGraphType qui nous permettaient de décrire les objets retournés par GraphQL. Ici nous allons créer des objets dérivant d'InputObjectGraphType.

Comme leur nom permet de le deviner, ces types permettent de décrire les objets d'entrée de nos mutations.

Voici un exemple demandant à fournir de manière obligatoire (encore une fois via l'utilisation de NonNullGraphType) le nom du Droïde à créer. L'identifiant n'est pas à fournir car il sera auto-généré par la base de données.

```
public class DroidInputType : InputObjectGraphType
{
  public DroidInputType()
  {
    Name = "DroidInput";
    Field<NonNullGraphType<StringGraphType>>("name");
  }
}
```

La mutation – définition de l'opération

```
public class StarWarsMutation : ObjectGraphType<object>
{
  public StarWarsMutation(StarWarsData data)
  {
    Name = "Mutation";
    Field<DroidType>(
      "createDroid",
      arguments: new QueryArguments(
       new QueryArgument<NonNullGraphType<DroidInputType>> {Name = "droid"}
      ),
      resolve: context =>
      {
        var droid = context.GetArgument<Droid>("droid");
        return data.AddDroid(droid);
      });
  }
}
```

Les mutations se déclarent de la même manière que les queries sous la forme de champs d'un objet ObjectGraphType.

Le GraphType passé en paramètre de la méthode Field permet de définir le type de retour de la mutation et la méthode resolve retourne l'objet nouvellement créé.

Le type donné dans l'argument est l'InputType que nous avons créé juste avant et qui permet de fournir les informations nécessaires à la création de l'objet.

La méthode GetArgument utilisée sur le contexte dans le « resolve » permet de récupérer notre input converti en POCO.

Ajout de la mutation dans le schéma

Maintenant que la mutation est prête il faut l'ajouter au schéma GraphQL déjà créé avec les queries. Pour cela on va renseigner dans son constructeur la propriété Mutation de la classe de base Schema dont hérite notre StarWarsSchema. On utilise encore une fois le IDependencyResolver d'Asp.net core pour le retrouver.

```
public class StarWarsSchema : Schema
{
  public StarWarsSchema(IDependencyResolver resolver)
    : base(resolver)
  {
    Query = resolver.Resolve<StarWarsQuery>();
    Mutation = resolver.Resolve<StarWarsMutation>();
  }
}
```

Exposition et utilisation via un service

Une fois cette étape réalisée, il ne reste plus qu'à modifier notre GraphQLService pour qu'il puisse exécuter une mutation. Pour cela on lui ajoute une méthode ExecuteMutationAsync qui utilise elle aussi un DocumentExecuter :

```
public class GraphQLService : IGraphQLService
{
  private readonly ISchema _query;
  public GraphQLService(ISchema query)
  {
    _query = query;
  }
  public Task<ExecutionResult> ExecuteQueryAsync(string query)
    => new DocumentExecuter()
      .ExecuteAsync(options =>
      {
        options.Schema = _query;
        options.Query = query;
      });
```

```
public Task<ExecutionResult> ExecuteMutationAsync(string query, string variables)
    => new DocumentExecuter()
        .ExecuteAsync(options =>
        {
            options.Schema = _query;
            options.Query = query;
            options.Inputs = variables.ToInputs();
        });
}
```

La méthode ToInput() est une méthode fournie par GraphQL permettant de parser le JSON des variables pour les convertir auto-magiquement en input GraphQL (décrit dans les objets InputObjectGraphType). Le service devra être appelé depuis l'endpoint d'un controller asp.net tel que celui-ci :

```
public async Task<IActionResult> DoMutationsAsync([FromBody] string query, [FromBody]
 string variables)
{
  ExecutionResult result = await _graphQLService.ExecuteMutationAsync(query, variables);

  return Ok(result);
}
```

Il ne reste plus qu'à tester notre implémentation en demandant la création d'un Droïde dans la base de données sous-jacente :

```
{
  "query": "mutation ($droid: DroidInput!){ createDroid(droid: $droid) { id name } }",
  "variables": {
    "droid": {
      "name": "R2-D2"
    }
  }
}
```

Le résultat que nous retourne l'API est alors le suivant : il s'agit de l'ExecutionResult converti en JSON par MVC.

```
{
  "data": {
    "droid": {
      "id": "AF669ABD-AAEB-4C7B-870D-0360FDFA02D5",
      "name": "R2-D2"
    }
  }
}
```

Gestion des erreurs

Comme nous l'avons vu plus haut, GraphQL à une très bonne gestion d'erreur lorsqu'il s'agit d'erreurs de syntaxe. Ces erreurs sont automatiquement générées par GraphQL.NET et ajoutées à la collection Errors de l'ExecutionResult.

Si une exception est levée lors de l'exécution d'une query, elle sera aussi automatiquement ajoutée à la collection Error dans une innerException. Cela n'est pas forcément très pratique à l'usage car on ne voit pas les innerExceptions dans le résultat JSON. Néanmoins, il est possible d'ajouter manuellement ses propres erreurs dans la collection Errors. Voici un exemple où l'on indique qu'un Droïde donné n'existe pas :

```
public class StarWarsQuery : ObjectGraphType<object>
{
  public StarWarsQuery(StarWarsData data)
  {
    Name = "Query";
    Field<DroidType>(
      "droid",
      arguments: new QueryArguments(
        new QueryArgument<NonNullGraphType<StringGraphType>>
        {
          Name = "id",
          Description = "id of the droid"
        }
      ),
      resolve: context =>
      {
        var droid = data.GetDroidById(id);
        if (data == null)
        {
          context.Errors.Add(new GraphQL.ExecutionError("Le droid n'existe pas.");
        }
        return droid;
    );
  }
}
```

Si le droid avec l'identifiant spécifié n'existe pas, on ajoute au context.Errors une nouvelle erreur de type ExecutionError. Voici le résultat reçu par le consommateur de l'API lors de l'exécution :

```
{
  "errors": [
    {
      "message": "Le droid n'existe pas."
    }
  ]
}
```

Conclusion

Ce qui est intéressant avec GraphQL c'est que vous pouvez ne l'utiliser que sur une partie bien spécifique de votre API et non pas forcément son intégralité. Vous pouvez n'exposer que certains controllers avec GraphQL et laisser les autres fonctionner de manière classique. Cela peut vous donc permettre également de migrer votre API sur GraphQL petit à petit. Il y a tout de même certains inconvénients avec GraphQL, notamment pour la mise en cache : comme les requêtes sont toutes différentes du fait de leur dynamisme, un cache ne peut pas être très utile devant une API GraphQL. Dans cet article, nous avons vu les bases de GraphQL, mais il faut savoir que cette technologie va beaucoup plus loin. Voici par exemple une liste non exhaustive de fonctionnalité très intéressantes qui n'ont pas été traitées dans cet article :

- Les fragments : permet de réutiliser plusieurs fois un bout de query à l'intérieur de la même query ;
- Les directives : permet d'ajouter ou d'ignorer les champs dans la query conditionnellement ;
- Les interfaces : fonctionne comme les interfaces en C# pour les ObjetGraphType ;
- Et beaucoup d'autres…

Il ne vous reste plus qu'à coder votre API GraphQL !

Jean Georges Perrin (@jgperrin)
*Architecte freelance dans le domaine des données (small and big data) et logiciel. Jean Georges a fondé et dirige plusieurs startups. Il a été le premier français (ex aequo) à être nommé IBM Champion en 2009 et l'est toujours. Il vit aujourd'hui en Caroline du Nord. Il est l'auteur de Spark with Java chez Manning (**https://www.manning.com/books/spark-with-java**).*

À la rencontre de CLEGO : le Cluster fabriqué en LEGO

L'été dernier, je me suis lancé, avec l'aide de mes fils, dans la fabrication d'un cluster pour étudier de façon plus approfondie le comportement d'Apache Spark. Avec des contraintes fortes sur le bruit, la température, les performances : je ne voulais pas reproduire les 4 IBM XSeries de 2U que j'avais acquis quelques années auparavant. Nous avons nommé ce projet CLEGO.

Bonjour CLEGO

CLEGO signifie Cluster pour L'analyse et l'Exploration des Giga Octets, ou plus simplement (et moins tiré par les cheveux), Cluster fabriqué avec des LEGO. 1

L'idée était de physiquement de construire un cluster silencieux, très puissant et compact pour étudier Apache Spark dans un environnement dédié, en dehors du cloud. Je crois beaucoup au cloud, mais le but ici est d'avoir un contrôle fin sur les ressources.

Ma première idée était d'essayer de trouver des maker boards comme le Raspberry Pi avec suffisamment de mémoire pour faire tourner Spark (au moins 8GO), mais, entre la rareté et les potentiels problèmes de compatibilité, je me suis dit qu'il valait mieux rester dans l'environnement x86.

Puis, je me suis dit que j'allais imprimer en 3D les différentes parties de CLEGO, mais le design allait me prendre des heures, sans compter les heures d'impression.

C'est à ce moment-là que mes fils et moi avons décidé de le faire en LEGO. Ce n'était pas non plus mon coup d'essai, il y a plus de 10 ans, j'avais construit un data center. La vidéo est encore sur YouTube (**https://youtu.be/e9Dbu9w5zEs**). La partie sur le data center commence à 1:41, mais soyez tolérant, ça date de 2006 !

Choisir le bon matos

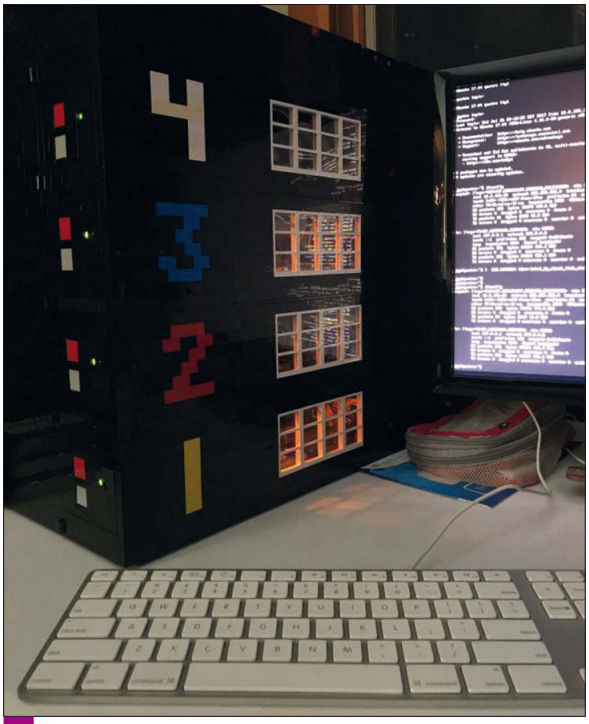
Ça commence avec le CPU (processeur). Je ne veux pas d'AMD (à tort peut être) et je ne peux pas me permettre des Xeon à cause de leur prix et des conséquences sur le reste du matériel : mémoire ECC, carte mère plus grande, etc. Je suis donc parti sur les i5-6400T (**https://ark.intel.com/products/88187/Intel-Core-i5-6400T-Processor-6M-Cache-up-to-2_80-GHz**). Ils ont 4 cœurs et 4 threads, tournent à 2.20 GHz, mais ce qui m'intéresse le plus, ils ne dissipent que 35W de chaleur (valeur du TDP - Thermal Design Power). Ce sont les versions des processeurs que l'on trouve dans les portables. La version sans le T, i5-6400 (**https://ark.intel.com/ products/88185/Intel-Core-i5-6400-Processor-6M-Cache-up-to-3_30-GHz**) dissipent 65W, près du double pour un peu plus de performance.

Maintenant que j'ai le CPU, il me faut une carte mère. J'ai fait quelques recherches. Au début, je voulais essayer de trouver une carte qui possèdent 2 connecteurs réseaux ou qui supportent du 10Gbits/s, mais j'ai renoncé, me disant que je pourrais toujours trouver un adaptateur USB3 vers Ethernet si nécessaire. J'ai finalement choisi Asus B150M-A/M.2 (**https://amzn.to/2kEmSFT**). Le format (form factor) est Micro ATX qui est une bonne dimension, mais je pouvais mettre 4 barrettes DDR4 et un « disque » M2.

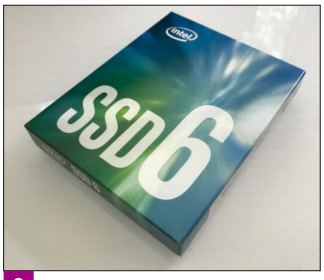
La mémoire n'a fait aucun doute pour moi. Je voulais de la DDR4 et remplir chaque nœud au maximum, soit 64 Go de RAM pour chacun. J'ai choisi la Corsair Vengeance LPX 32Go (2x16Go) DDR4 2133 MHz (**https://amzn.to/2ITcQzk**). Même si je n'avais pas beaucoup de doute, cette ligne du budget a été la plus difficile à avaler ! Mais bon, au final, mon cluster a 256 Go de RAM, c'est la taille du SSD de mon portable du bureau ! 2 3

Comme le niveau sonore était une réelle priorité pour moi, je ne voulais pas de disque dur mécanique. Au début, je pensais à des SSD 2,5 pouces, mais j'ai opté pour des « disques » m2. Après un peu de recherches, je suis parti sur des Intel SSD6 (**https://amzn.to/2sxiOLi**) qui sont du stockage NVMe avec des performances exceptionnelles : lecture séquentielle jusqu'à 1775 Mo/s, écriture séquentielle jusqu'à 560 Mo/s et 128,5KIOPS. Il est vrai que ce sont les performances en accès séquentiel et Spark ne va probablement pas écrire que séquentiellement, mais ne chipotons pas…

Malgré la faible dissipation thermique de mes processeurs, je voulais un système dynamique de refroidissement par air. J'avoue que le refroidissement par eau m'a paru tentant, mais je suis resté sur l'air et j'ai choisi le ventilateur Cooler Master GeminII M4 (**https://amzn.to/2LPzLtf**), pour sa compacité, le fait qu'il ait quatre caloducs en cuivre et un grand ventilateur à vitesse variable. Encore une fois, je cherche à diminuer le niveau sonore. Je dois avouer que j'ai été assez



1 *Ubuntu 16 sur un des nœuds de CLEGO.*



2 *CLEGO utilise du stockage NVMe, comme les serveurs et portables haut de gamme.*

3 *Le stockage NVMe ressemble beaucoup aux modules mémoires, à la différence, les connecteurs sont sur le « petit » côté.*

4

Le thermostat de CLEGO affiche la température de la sonde et la température désirée… en degrés Fahrenheit (110 °F = 43.3 °C, 86 °F = 30 °C).

impressionné par le packaging de Cooler Master et l'ingéniosité des pièces mécaniques. Du côté de l'alimentation électrique, je me suis dit que je pourrais construire CLEGO qu'avec deux alimentations, avec un nombre de watts plus élevés plutôt qu'une alimentation par carte mère. Je me suis dirigé vers des SeaSonic SS-520FL2 (https://amzn.to/2OqDhvt). Elles n'ont pas de ventilateurs et offrent un câblage complètement modulaire : on ne branche que ce dont on a besoin et on n'est donc que peu encombré par une flopée de câbles qu'on ne va jamais utiliser ! Chaque bloc d'alimentation va uniquement alimenter deux cartes mères et deux processeurs, du coup, un minimum de place est pris par les câbles. Par contre, chaque alimentation ne peut alimenter qu'une carte mère, j'ai donc dû acheter un câble en Y pour diviser l'alimentation en deux : j'ai trouvé ça chez Eyeboot. Le câble s'appelle ATX 24-Pin female to 24-pin male Y splitter power câble (https://amzn.to/2LtBVCp).

Le réseau est assuré par un switch TOR (top of rack, ou, au-dessus du rack). Il consiste en un simple, mais efficace switch 8 ports, manageable Netgear GS108Ev3 (https://amzn.to/2v0Dq0p). Au début, je voulais quatre ports pour mes quatre nœuds et un port pour la sortie. J'avais pensé à un port pour un Raspberry Pi pour monitorer l'environnement (température, humidité, etc.), mais j'ai finalement laissé tomber cette idée de service supplémentaire, un switch cinq ports aurait été suffisant. J'ai également un petit convertisseur femelle/femelle RJ45 (https://amzn.to/2v16eGG) qui me permet d'avoir une prise femelle sur le boîtier.

En parlant d'équipement ancillaire, j'ai également inclus un KVM 4 ports de chez logear (GCS24U, https://amzn.to/2K4si7F). Il est un peu grossier, surtout avec tous les câbles, mais j'aime leur idée de petite télécommande (à fil), comme cela, je n'ai pas besoin de prévoir un accès pour le contrôle. 4

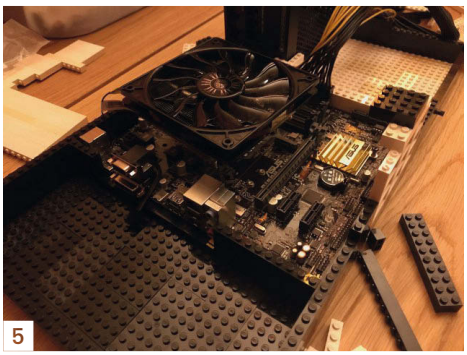
Malgré la faible température des processeurs et leur ventilateur, je voulais être sûr que l'air chaud ne reste pas sur les cartes mères. Bien que les deux alimentations n'aient pas de ventilateur, les calories dégagées doivent toujours être dissipées ! J'ai ajouté un système de ventilation AC Infinity Airplate T7 (https://amzn.to/2mSid4z). Ce système est typiquement utilisé dans les meubles faits pour ranger les équipements audiovisuels pour un home cinema. Encore une fois, je recherche le silence…

Finalement, j'ai utilisé quelques petits câbles pour les LED, l'allumage des PC et le bouton reset (https://amzn.to/2mS46vR) ainsi que des petits haut-parleurs, qui servent, toujours, à débugger une carte mère (https://amzn.to/2NUwhG0).

Voilà… nous sommes prêts pour l'assemblage !

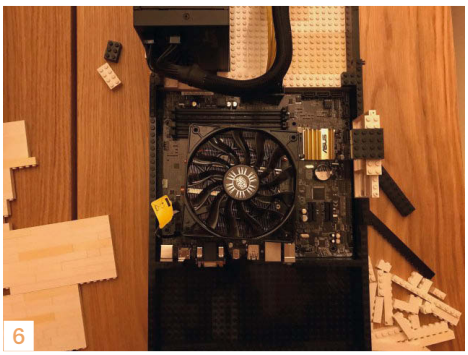
CONSTRUIRE LE CHÂSSIS DE CLEGO

La construction a pris le plus de temps : essai, démontage, remontage, achat de pièces à l'unité, etc.



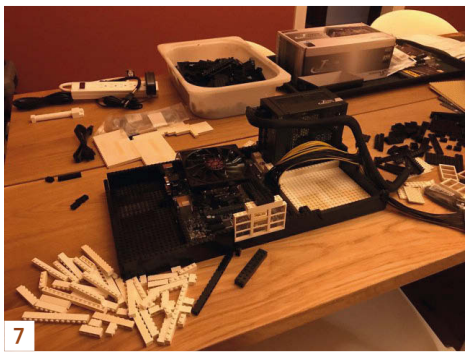
5

J'ai décidé de maintenir l'empreinte du système sur l'équivalent de deux plaques de base, soit 32 par 64 tenons. Vous pouvez voir les connecteurs de la carte mère au premier plan et la première alimentation à l'arrière-plan.



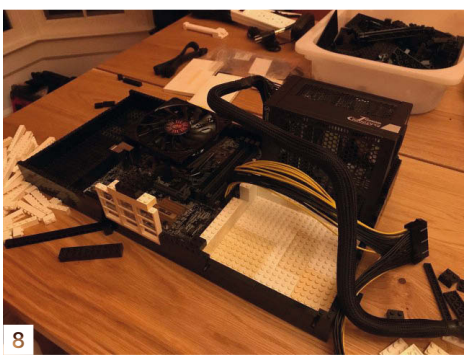
6

Une vue aérienne du projet : à cette étape, je mesurais encore les éléments. Le processeur n'est pas dans son socle ni la mémoire. Le ventilateur est juste posé (avec ses protections) sur la carte mère.



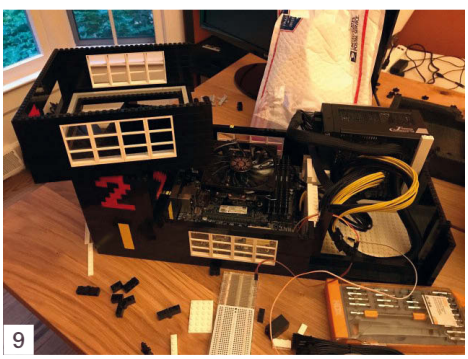
7

Une vue plus globale de ma table de salle à manger, qui s'est avérée être un précieux atout dans cette expérience (j'ai aussi mangé ailleurs pendant la durée du projet…)



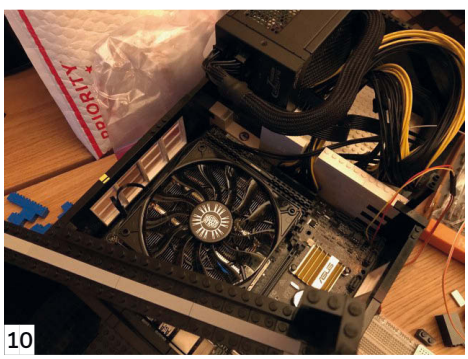
8

Vue détaillée de la partie alimentation, que je voulais blanche (white room). Au début j'avais pensé à trois fenêtres de chaque côté, mais le projet final en comportera quatre.



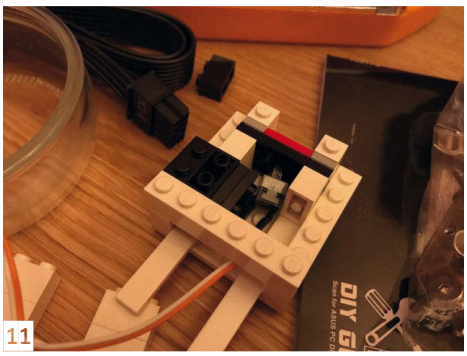
9

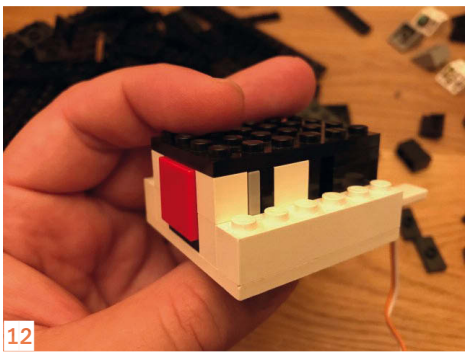
Démarrage du deuxième nœud. À noter que les nœuds 2, 3 et 4 ont chaque fois leur petit caddy, qui devrait permettre une maintenance plus aisée si le besoin se faisait sentir.

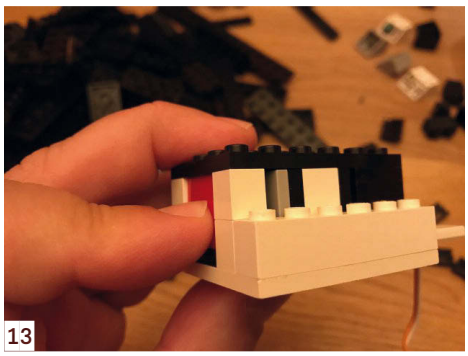


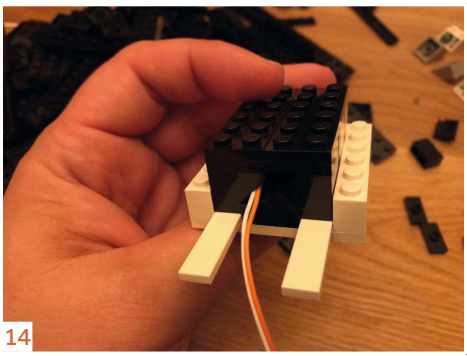
10

Le premier nœud est prêt : la carte mère est équipée du processeur, de la mémoire et du disque NVMe que l'on voit sous le ventilateur.

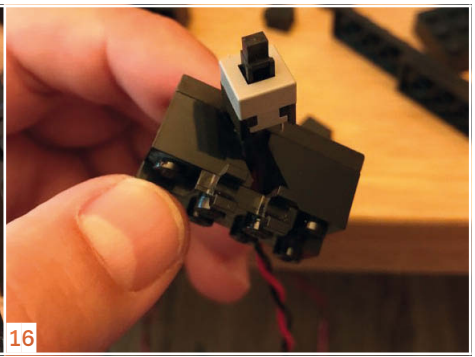

11
Début des travaux sur l'interrupteur pour l'alimentation.

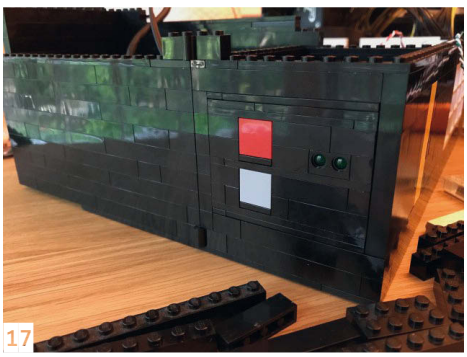

12
Première tentative d'utilisation de l'interrupteur et…

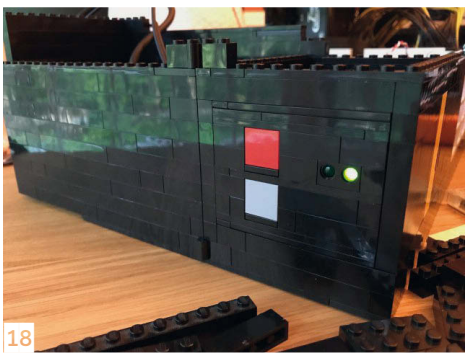

13
Ça marche !

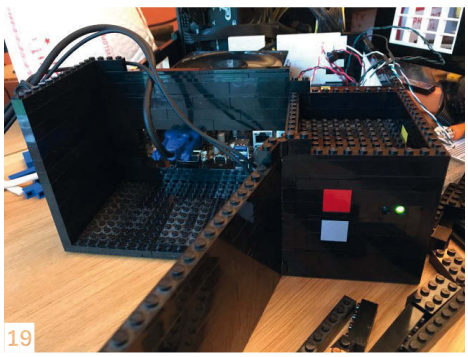

14
Faites bien attention que les câbles soient un peu lâches, mais pas l'interrupteur en lui-même.

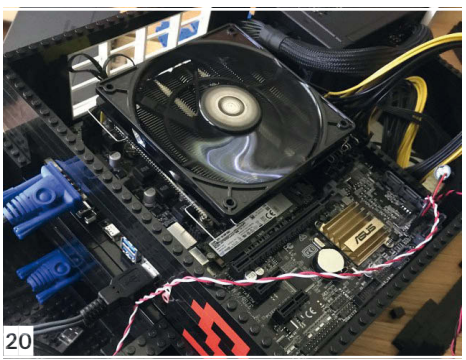

15
Zoom sur le bouton reset. Ne faites pas attention à la brique qui s'est perdue…

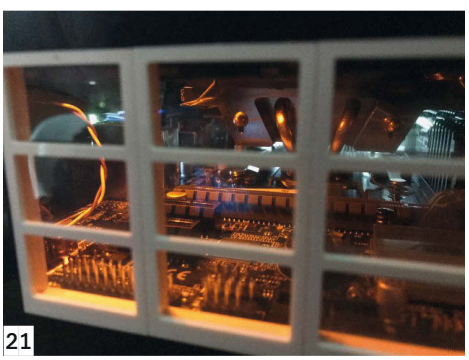

16
Zoom sur le switch.

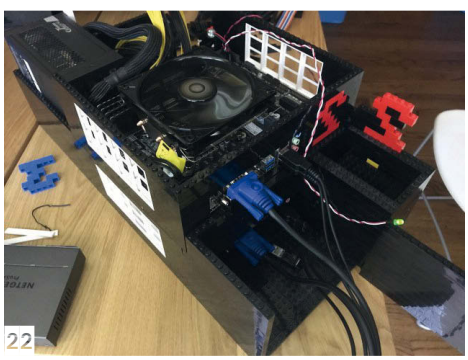

17
ça y est, le premier nœud est prêt.


18
Et le premier nœud fonctionne…

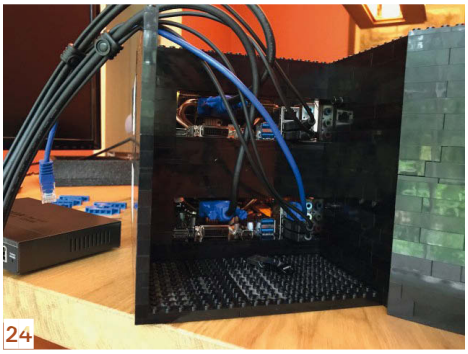

19
Porte d'accès aux connecteurs de la carte mère.


20
Le deuxième nœud fonctionne.

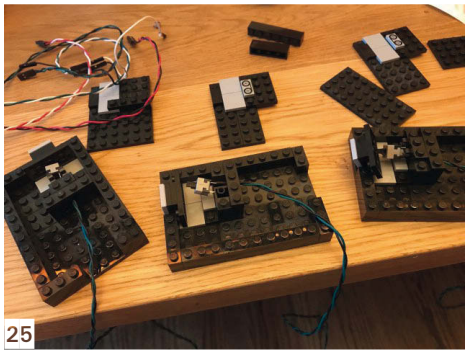

21
Vue des composants au travers des fenêtres.


22
Les deux premiers nœuds, fonctionnant de façon indépendante, mais utilisant la même alimentation.

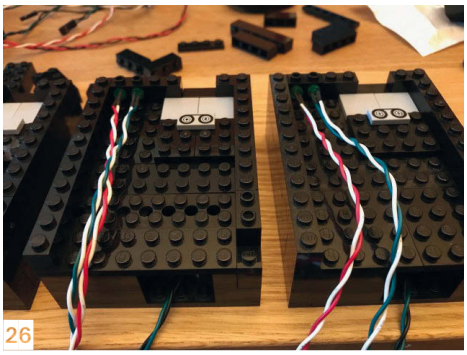
Vue sur le BIOS d'Asus en arrière-plan.



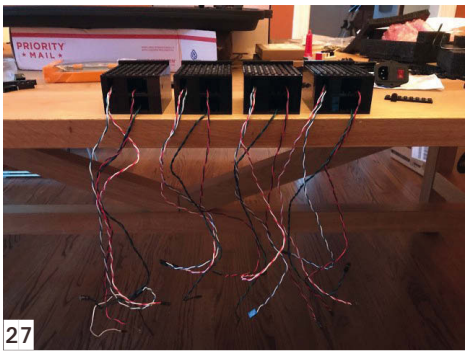
Le câblage frontal commence à être un peu désorganisé avec deux nœuds (et sans les câbles réseaux). Qu'est-ce que cela va être avec quatre ?



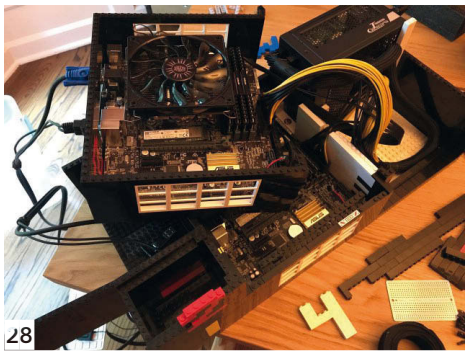
Les interrupteurs et les LEDs sont montés dans un petit conteneur qui glissera dans le châssis, pour faciliter la maintenance.



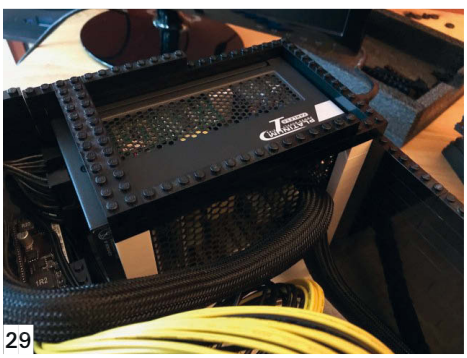
Taylorisme en action.



Les quatre blocs de contrôle sont prêts avec les câbles pour l'interrupteur d'alimentation, le bouton reset, la LED d'alimentation et la LED disque.



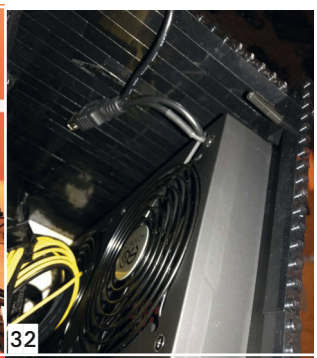
Chaque nœud, à l'exception du premier, est dans une cage. Chaque cage se fixant sur l'autre comme les modèles Creator Expert de LEGO (voir https://shop.lego.com/fr-FR/Un-diner-au-centre-ville-10260)



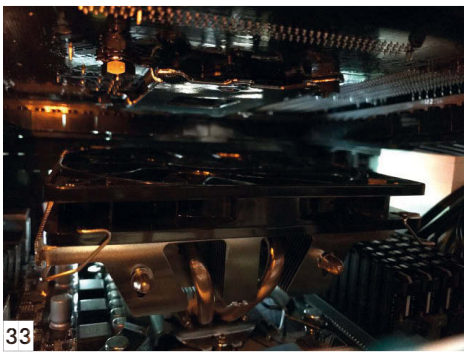
Zoom sur le bloc d'alimentation.

La deuxième alimentation est en place. Au premier plan, le double ventilateur est en train d'être sécurisé.

Le ventilateur est légèrement encastré pour ne pas avoir de pièces qui dépassent. Sur la gauche, vous pouvez voir l'espace libre pour les nœuds 3 et 4.

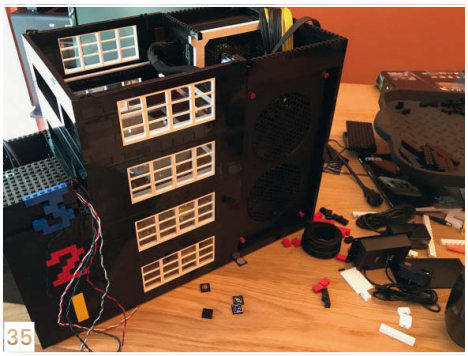


À l'intérieur de la tour de refroidissement. À un moment, je voulais mettre Luke Skywalker et Dark Vador avec leurs sabres laser, mais les câbles ont pris tout l'espace.

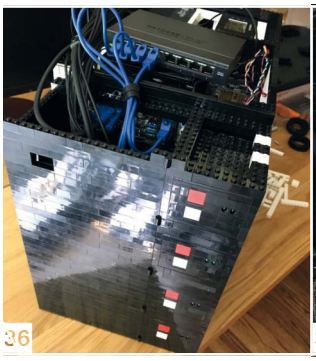


Inspection de l'espace entre le haut d'un nœud et le bas d'un autre.

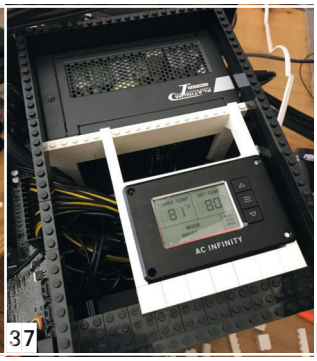
Les cages des nœuds 3 et 4 en préparation. Ne faites pas attention à la colle (Kragle) au fond. Aucune colle n'a été utilisée dans ce projet.



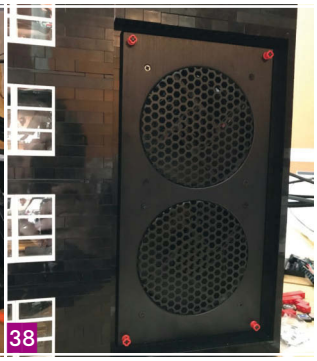
Enfin une idée du volume final.


36

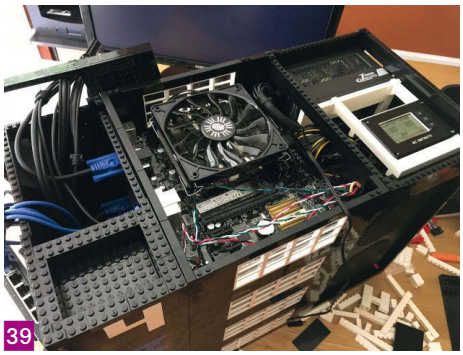
Préparation du switch « top of rack ». Vous pouvez aussi voir, au premier plan, la porte du châssis avec son verrou.


37

Intégration du thermostat, vous pouvez distinguer le câble noir sur la gauche qui emmène la sonde au plus près du processeur.


38

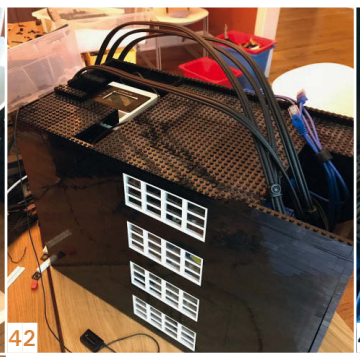
Les ventilateurs externes assemblés.


39

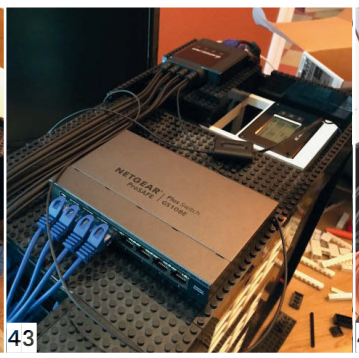
Les quatre nœuds assemblés, en attente du couvercle.


41

Les câbles sont pénibles…


42

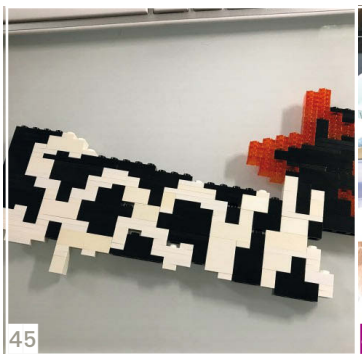
Le toit est prêt pour l'installation des composants communs.


43

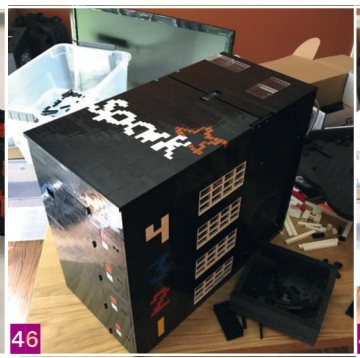
Intégration du switch et du KVM, vous pouvez distinguer la télécommande du KVM entre le thermostat et le switch.


44

Détails sur le top of rack : le câble orange part vers le réseau, les câbles bleus partent vers chaque nœud. Au fond, les câbles du KVM.


45

Travail sur le logo en LEGO.


46

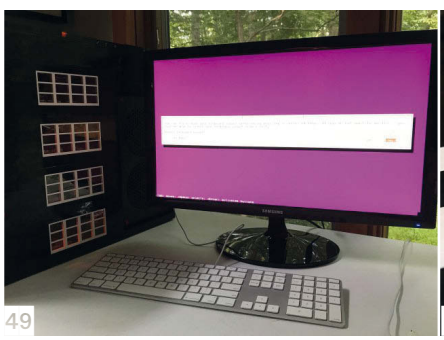
Assemblage terminé avec le couvercle fermé.


47

CLEGO avec son couvercle avant ouvert.


48

CLEGO peut être transporté, mais j'avoue que ça n'est pas mon exercice favori.


49

Installation d'Ubuntu sur chaque nœud… un peu fastidieux, mais je prends le rythme.


50

Logo final.

Et au final…

CLEGO mesure environ 25,4 cm x 50,8 cm x 42,2 cm. Il est désormais sur un support de moniteur et je peux l'avoir sous mon bureau, comme un gros desktop. Je n'ai pas encore pesé la bête, il faudrait que je le fasse à l'occasion, mais elle est assez lourde. Déplacer CLEGO est toujours générateur de stress.

Fabriquer CLEGO a été un projet plutôt amusant, complètement hors budget, complètement hors des délais (c'est pour cela que j'ai dû finir seul sans mes fils). Mais il faut savoir s'écarter du quotidien de temps en temps, non ?

Vous trouverez quelques photos supplémentaires sur mon blog, à http://jgp.net/2017/09/22/new-dimension-apache-spark-clusters/.

Denis Duplan,
sociologue et développeur à ses heures.
Blog : http://www.stashofcode.fr

Afficher des sprites et des BOBs sur Amiga OCS et AGA

Partie 1

niveau 200

Quoi de plus confortable qu'un sprite ? Le coprocesseur graphique l'affiche en laissant voir le décor derrière ses pixels transparents, et il préserve le décor derrière ses autres pixels pour le restaurer quand le sprite est déplacé. Par ailleurs, il découpe le sprite s'il sort de l'écran.

Malheureusement, les capacités de l'Amiga 500 sont en la matière très limitées. Huit sprites de 16 pixels de large, même de hauteur infinie, en 4 couleurs dont une transparente, seulement ? C'est la dèche…

Les sprites n'en conservent pas moins une certaine utilité, à condition d'en utiliser pleinement le potentiel. Ainsi, il est possible de les attacher pour former jusqu'à un bitmap de 64 pixels de large, de hauteur infinie, en 16 couleurs dont une transparente, notamment. Ou alors, il est possible d'en réutiliser pour créer un playfield supplémentaire, dont le contenu devra toutefois être un motif répétitif qui, dans le meilleur des cas, occupera 48 pixels de large sur une hauteur infinie, en peu de couleurs. Par ailleurs, l'*Advanced Graphics Architecture*, dont l'Amiga 1200 est doté, dispose de fonctionnalités un peu étendues en matière de sprites. En particulier, leur largeur unitaire passe de 16 à 64 pixels.

Comment afficher des sprites ? Et utiliser les sprites pour afficher un gros et beau bitmap, ou un playfield supplémentaire ? Et enfin utiliser les sprites sur AGA, sachant que Commodore n'a jamais documenté ce hardware ? Tout cela et plus encore dans ce premier article consacré à l'affichage de bitmaps sur Amiga OCS et AGA.

Vous pouvez télécharger l'archive contenant le code et les données des programmes présentés ici à l'URL suivante :

http://www.stashofcode.fr/code/afficher-sprites-et-bobs-sur-amiga/code.zip

Cette archive contient plusieurs sources :

- spriteCPU.s pour le simple affichage d'un sprite au CPU (ie : sans DMA) ;
- sprites.s pour le simple affichage et le déplacement d'un sprite ;
- sprites16.s pour le simple affichage et le déplacement d'un sprite en 16 couleurs ;
- spritesField.s pour la réutilisation de sprites comme pour un fond étoilé ;
- spritesCollision.s pour la détection de collisions entre deux sprites, et entre ces sprites et un bitplane ;
- triplePlayfield.s pour l'affichage d'un plan de sprites (avec dual-playfield tant qu'on y est) ;
- spritesAGA.s pour l'affichage de sprites exploitant le chipset AA.

Comme toujours depuis qu'il en est question dans ces colonnes, vous pouvez pratiquer la programmation sur Amiga à l'aide de l'assembleur ASM-One tournant sur une émulation d'Amiga 500 ou 1200 dans WinUAE, en vous appuyant au besoin sur l'*Amiga Hardware Reference Manual*. Vous trouverez les références à la fin de cet article.

Référez-vous au premier article de la série portant sur la programmation d'un sine scroll (*Programmez !* #213 à #217), pour plus d'informations. Alternativement, vous pouvez vous reporter au blog de l'auteur.

Note

Cet article se lit mieux en écoutant l'excellent module composé par Spirit / LSD pour le diskmag Graphevine #14, mais c'est affaire de goût personnel…

Le B-A-BA : afficher un sprite, pour commencer

Les bases concernant les sprites ont déjà été très brièvement présentés dans un article portant sur la programmation d'une cracktro sur Amiga (*Programmez !* #211). Brève piqûre de rappel, donc, et compléments intensifs pour montrer comment utiliser précisément les fonctionnalités des sprites.

L'Amiga dispose d'un coprocesseur graphique nommé Copper. Le Copper exécute une série d'instructions à chaque rafraîchissement de l'écran. Ces instructions sont notamment des écritures dans des registres du hardware, dont certains contrôlent les sprites.

Le hardware peut afficher huit sprites de 16 pixels de large en 4 couleurs dont une couleur transparente, sur une hauteur illimitée. Les sprites sont couplés (le 0 avec le 1, le 2 avec le 3, etc.). Dans un couple, les sprites partagent la même palette de 4 couleurs, qui n'est qu'un sous-ensemble de la palette de 32 couleurs utilisée pour afficher des pixels à l'écran. Dans ces conditions, la structure de cette dernière palette est la suivante (la mention aux playfields sera expliquée plus tard) :

Couleurs	Usage
00 à 07	Playfield 1 (bitplanes 1, 3 et 5)
08 à 15	Playfield 2 (bitplanes 2, 4 et 6)
16 à 19	Sprites 0 et 1
20 à 23	Sprites 2 et 3
24 à 27	Sprites 4 et 5
28 à 31	Sprites 6 et 7

Un sprite est intégralement décrit par une suite de mots. Les deux premiers sont des mots de contrôle, les suivants sont des mots décrivant ligne par ligne de petits bitplanes, et les derniers sont des 0. Par exemple, pour un sprite de deux lignes :

```
sprite:   DC.W $444C, $5401     ;Mots de contrôle
          DC.W $5555, $3333     ;Première ligne de 16 pixels
          DC.W $1234, $5678     ;Seconde ligne de 16 pixels
          DC.W 0, 0             ;Fin
```

Pour chaque ligne du sprite, les deux mots sont combinés entre eux pour en déduire les indices des couleurs des 16 pixels de la ligne.

Pour poursuivre sur cet exemple, le résultat pour la première ligne est celui représenté en Figure 1. Ainsi qu'il est possible de le constater, le premier mot de la ligne fournit les bits 0 de la ligne, et le second mot en fournit les bits 1.

Les coordonnées (X, Y) et la hauteur d'un sprite (déduite de l'ordonnée Y + DY de la ligne qui suit la dernière ligne du sprite) sont codées dans les mots de contrôle de manière assez exotique (le bit 7 du second mot est réservé pour l'attachement, ce dont il sera question plus loin). Par exemple, pour afficher un sprite en haut à gauche de l'écran qui débute en classiquement en ($81, $2C), comme expliqué en Figure 2.

Oui, contrairement à ce que prétend l'*Amiga Hardware Reference Manual*, il faut bien coder `X-1` et non `X` dans les mots de contrôle. C'est une erreur de la documentation.

Concrètement, il faut donc coder les mots de contrôle ainsi (dans ASM-One, le point d'exclamation est un `OR`, et `>>` est un décalage non signé sur la droite) :

```
sprite:     DC.W ((Y&$FF)<<8)!(((X-1)&$1FE)>>1)
            DC.W (((Y+DY)&$FF)<<8)!((Y&$100)>>6)!(((Y+DY)&$100)>>7)!((X-1)&$1)
```

Pour s'éviter de tels calculs chaque fois qu'il faut déplacer un sprite, une technique consiste à précalculer des mots à combiner pour chacune des 320 positions horizontales, d'une part, et chacune des positions verticales d'autre part. Pour en savoir plus, référez-vous à ce fil d'un forum de l'English Amiga Board :
http://eab.abime.net/showthread.php?t=81835.

La suite des mots constituant les données d'un sprite étant écrites, il suffit d'en communiquer l'adresse (qui doit être paire) au hardware pour qu'il affiche le sprite. Chaque sprite dispose de registres `SPRxPTH` et `SPRxPTL` pour cela. Par exemple :

```
lea #dff000,a5
move.l #sprite,d0
move.w d0,SPR0PTL(a5)       ;$DFF122
swap d0
move.w d0,SPR0PTH(a5)       ;$DFF120
```

Dans les faits, les adresses des sprites ne sont jamais communiquées ainsi par le CPU. Des instructions MOVE sont rajoutées dans la Copper list pour que le Copper les communique à chaque trame :

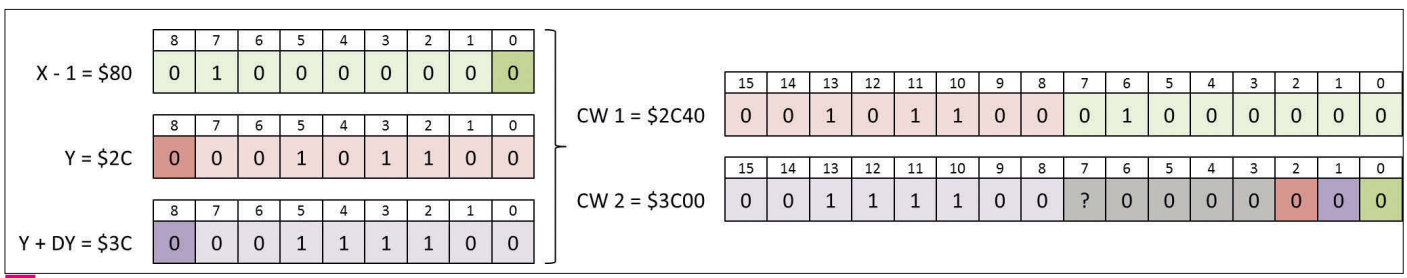
```
lea copperList,a0
;... (début de la Copper list)
move.l #sprite,d0
move.w #SPR0PTL,(a0)+
move.w d0,(a0)+
swap d0
move.w #SPR0PTH,(a0)+
move.w d0,(a0)+
;... (suite de la Copper list)
```

Les sprites ne sont affichés que si le hardware peut accéder à leurs données, ce pourquoi il bénéficie d'un accès direct en mémoire (DMA). L'usage du DMA n'est pas incontournable – il est possible de s'y substituer en écrivant au CPU dans les divers registres où le DMA écrit les données des sprites pour les communiquer au hardware : `SPRxPOS`, `SPRxCTRL`, `SPRxDATB` et `SPRxDATA`. Le programme `spriteCPU.s` procède ainsi :

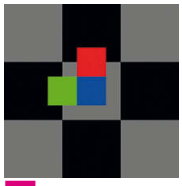
```
;Attendre la première ligne du sprite pour commencer à l'afficher

_waitSpriteStart:
   move.l VPOSR(a5),d0
   lsr.l #8,d0
   and.w #$01FF,d0
   cmpi.w #SPRITE_Y,d0
   blt _waitSpriteStart

   ;Afficher le sprite. Il faut écrire dans SPRxDATA en dernier, car c'est le moyen de
déclencher l'affichage du sprite.

   move.w #((SPRITE_Y&$FF)<<8)!(((SPRITE_X-1)&$1FE)>>1),SPR0POS(a5)
   move.w #(((SPRITE_Y+SPRITE_DY)&$FF)<<8)!((SPRITE_Y&$100)>>6)!(((SPRITE
_Y+SPRITE_DY)&$100)>>7)!((SPRITE_X-1)&$1),SPR0CTL(a5)
   move.w #$0F0F,SPR0DATB(a5)
   move.w #$00FF,SPR0DATA(a5)

   ;Attendre la ligne de milieu du sprite pour le repositionner horizontalement (8 pixels
plus à droite) et modifier ses données

_waitSpriteMiddle:
   move.l VPOSR(a5),d0
```

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1er mot	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
2nd mot	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Offset	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
Couleur	16	17	18	19	16	17	18	19	16	17	18	19	16	17	18	19
Pixels																

1 *Combinaison des mots d'une ligne de données du sprite 0.*



2 *Codage des mots de contrôle d'un sprite de DY pixels de hauteur affiché en (X, Y).*



3

Le sprite 0, en 4 couleurs, circulant paisiblement sur un bitplane.

```
    lsr.l #8,d0
    and.w #$01FF,d0
    cmpi.w #SPRITE_Y+(SPRITE_DY>>1),d0
    blt _waitSpriteMiddle
    move.w #((SPRITE_Y&$FF)<<8)!(((SPRITE_X+8-1)&$1FE)>>1),SPR0POS(a5)

    ;Ecrire dans SPRxCTL arrête l'affichage du sprite, ce qui interdit de repositionner le sprite horizontalement à la précision d'un pixel, à moins de le réarmer par une écriture dans SPRxDATA, car le bit 0 de cette position est dans SPRxCTL. Autrement dit, les trois lignes qui suivent ne sont nécessaires que si la nouvelle position horizontale est impaire.

    move.w #(((SPRITE_Y+SPRITE_DY)&$FF)<<8)!((SPRITE_Y&$100)>>6)!(((SPRITE_Y+SPRITE_DY)&$100)>>7)!((SPRITE_X+9-1)&$1),SPR0CTL(a5)
    move.w #$F0F0,SPR0DATB(a5)
    move.w #$FF00,SPR0DATA(a5)

    ;Attendre la dernière ligne du sprite pour cesser de l'afficher en écrivant n'importe quoi dans SPRxCTL.

_waitSpriteEnd:
    move.l VPOSR(a5),d0
    lsr.l #8,d0
    and.w #$01FF,d0
    cmpi.w #SPRITE_Y+SPRITE_DY,d0
    blt _waitSpriteEnd
    move.w #$0000,SPR0CTL(a5)
```

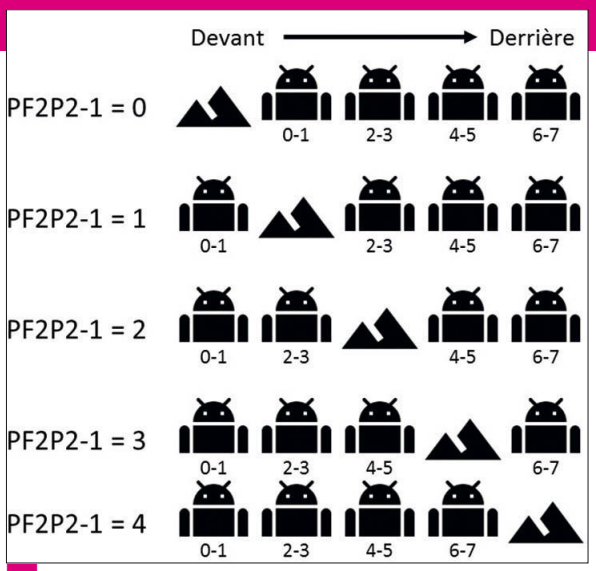
Toutefois, cette technique présente peu d'intérêt, sinon aucun. C'est que pour afficher un sprite, il faudrait donc écrire une boucle qui attendrait le faisceau d'électron à chaque ligne à laquelle une ligne du sprite devrait être affichée avant de modifier le contenu des registres `SPRxDATB` et `SPRxDATA` avec les données de la nouvelle ligne du sprite, ce qui serait extrêmement contraignant.
Le DMA n'est pas émulé, mais utilisé. Pour afficher les sprites, il est donc nécessaire d'activer au moins les canaux DMA du Copper, des bitplanes et des sprites :

```
    move.w #$83A0,DMACON(a5)     ;DMAEN=1, BPLEN=1, COPEN=1, SPREN=1
```

Comme cela peut le suggérer, il n'existe pas de mécanisme pour activer sélectivement tel ou tel des huit sprites. Pour ne pas afficher un sprite, il faut spécifier au hardware que la hauteur du sprite est nulle, ce qui s'effectue en faisant pointer ses données des mots de contrôle à 0 :

```
spriteVoid:     DC.W 0, 0
```

Tant qu'il est question de DMA, un point de détail, mais qui a tout de même son importance, doit être mentionné. Il faut attendre un blanc vertical pour couper le canal DMA des sprites. A défaut, si un sprite était en cours d'affichage, ses données continuent d'être affichées. Cela produit un effet bien connu de « sprite qui bave verticalement ».
En effet, comme expliqué précédemment lorsqu'il a été question de l'émuler au CPU, le DMA ne sert qu'à alimenter des registres `SPRxPOS` , `SPRxCTL` , `SPRxDATA` et `SPRxDATB` dans lesquels le hardware lit systématiquement les données des sprites qu'il combine à celle des bitplanes. S'il est coupé avant d'avoir écrit le dernier jeu de mots à 0 dans `SPRxPOS` et `SPRxCTL` (ce qu'il fait lorsqu'il sait que la hauteur du sprite a été parcourue, ce qui explique pourquoi ces deux mots doivent figurer à la fin des données d'un sprite), le DMA ne peut donc interrompre l'affichage du sprite. Ce dernier continue d'être affiché sur tout le reste de la hauteur de l'écran avec les données figurant dans `SPRxDATA` et `SPRxDATB` au moment où le DMA a été coupé.
C'est pourquoi les programmes proposés attendent le blanc vertical (VERTB), c'est-à-dire le moment où le faisceau d'électrons a terminé de tracer l'écran, pour couper le DMA (la boucle est factorisée dans la sous-routine `_waitVERTB`) :



4 *Ah ! Je vous ai bien eu. Cet article traite d'Amiga et non d'Androïd !*

```
_waitVERTBLoop:
    move.w INTREQR(a5),d0
    btst #5,d0
    beq _waitVERTBLoop
    move.w #$07FF,DMACON(a5)
```

La cinétique : déplacer les sprites et détecter les collisions

Pour déplacer un sprite, il suffit de modifier ses coordonnées dans ses mots de contrôle à la fin d'une trame, avant que le hardware n'en refasse la lecture pour afficher la nouvelle trame. Le programme sprite.s fait ainsi se déplacer un sprite 0 de 16 pixels de hauteur en 4 couleurs, sur un décor composé d'un unique bitplane représentant un damier (Figure 3).
Le hardware permet de gérer la profondeur via un système de priorités :

- entre sprites d'abord, les priorités sont figées : le sprite 0 est toujours affiché devant le sprite 1, qui est toujours affiché devant le sprite 2, etc ;
- entre sprites et playfields ensuite (pour rappel, le hardware peut afficher un playfield de 1 à 6 bitplanes, ou deux playfields de 1 à 3 bitplanes chacun – c'est le dual-playfield), les priorités peuvent être ajustées via le registre `BPLCON2` .

`BPLCON2` se compose ainsi :

Bits	Usage
15 à 7	Inutilisé (mettre à 0)
6	PF2PRO
5 à 3	PF2P2 à PFR2P0
2 à 0	PF1P2 à PF1P0

Dans le cas d'un seul playfield, PF2P2-0 (et non PF1P2-0, comme on l'aurait présumé) sont utilisés. Ils permettent de coder sur 3 bits le numéro du couple de sprites (et non du sprite !) derrière lequel se trouve le playfield : 4

En dual-playfield, PF1P2 à PF1P0 permettent de gérer pareillement la priorité du second playfield et des couples de sprites. Noter que ces bits traitent alors bien du playfield 1 (bitplanes 1, 3 et 5) tandis que PF2P2-0 traitent du playfield 2 (bitplanes 2, 4 et 6). Le bit PF2PRO permet de passer le playfield 2 devant le playfield 1, ce qui rajoute aux possibilités de priorisation entre sprites et playfields.

Dans les exemples déjà cités des sprites en 4 et 16 couleurs, le sprite 0 est positionné devant l'unique playfield en stockant donc `$0008` dans `BPLCON2`.

Enfin, il faut mentionner que le hardware permet de détecter facilement une collision entre sprites, ou entre sprite et bitplanes, et cela au pixel près. Le programme `spritesCollision.s` montre comment il est possible de détecter les collisions entre le sprite 0 et le sprite 2, et entre chacun de ces sprites et le bitplane 1 (Figure 5).

La détection des collisions est assez subtile. Elle repose sur l'emploi de deux registres :

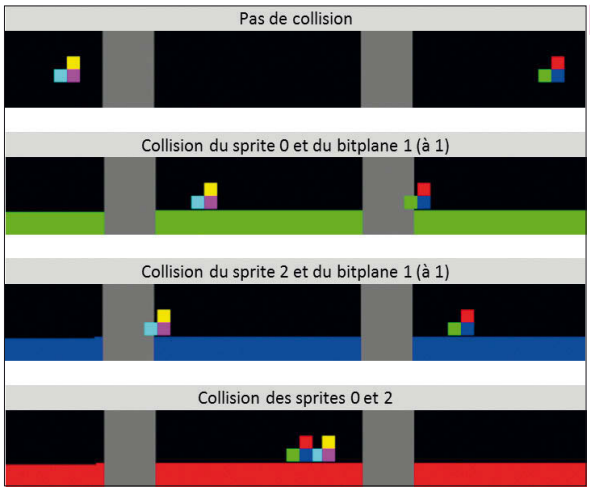
- `CLXCON` permet de spécifier quels sprites et quels bitplanes impliquer ;
- `CLXDAT` permet de récupérer l'état de la détection de collisions qui en résulte.

La détection de collisions implique toujours les sprites pairs. Dans `CLXCON` :

- Les quatre bits ENSP7-1 permettent de spécifier que les sprites impairs doivent aussi être impliqués. Noter que cela ne permet pas de détecter distinctement les collisions du sprite 0 et les collisions du sprite 1, mais de détecter les collisions de la combinaison par OR des sprites 0 et 1 : ce n'est donc vraiment utile que si ces sprites sont attachés pour afficher un sprite en 16 couleurs.
- Les six bits ENBP6-1 permettent de spécifier si les bitplanes correspondants doivent être impliqués (si tous sont exclus, une collision avec les bitplanes est systématiquement rapportée, quelles que soient les valeurs des bits MVBP6-1).
- Les six bits MVBP6-1 permettent de spécifier la valeur du bit dans un bitplane impliqué dans la détection de collisions qui permettra de déterminer s'il y a ou non collision avec ce bitplane.

Dans le programme `spritesCollision.s`, la valeur de `CLXCON` est `$0041`, ce qui revient à positionner le bit ENBP1 pour impliquer le bitplane 1 dans la détection de collisions, et le bit MVBP1 pour spécifier qu'une collision avec le bitplane 1 a lieu quand un bit à 1 est rencontré dans ce dernier.

Il suffit de lire `CLXDAT` (attention, car cela le remet à 0) pour récupérer à chaque trame l'état des collisions détectées entre deux entités (entre sprites, ou entre sprite et bitplane, ou entre bitplanes). Sur la figure (Figure 6), on trouve à côté de chaque bit une case par sprite (0 à 7) ainsi qu'une case pour les bitplanes impairs (I) et



5 *Détection de collisions entre sprites et entre sprites et bitplane.*

15	Inutilisé									7
14	0	1	2	3	4	5	6	7	I	P
13	0	1	2	3	4	5	6	7	I	P
12	0	1	2	3	4	5	6	7	I	P
11	0	1	2	3	4	5	6	7	I	P
10	0	1	2	3	4	5	6	7	I	P
9	0	1	2	3	4	5	6	7	I	P
8	0	1	2	3	4	5	6	7	I	P

7	0	1	2	3	4	5	6	7	I	P
6	0	1	2	3	4	5	6	7	I	P
5	0	1	2	3	4	5	6	7	I	P
4	0	1	2	3	4	5	6	7	I	P
3	0	1	2	3	4	5	6	7	I	P
2	0	1	2	3	4	5	6	7	I	P
1	0	1	2	3	4	5	6	7	I	P
0	0	1	2	3	4	5	6	7	I	P

6 *La composition du registre CLXDAT.*

une pour les bitplanes pairs (P). Les cases sont colorées pour indiquer la collision détectée (première entité en rouge, seconde en vert). Par exemple, le bit 7 sera positionné si une collision est détectée entre :

- le sprite 4, éventuellement combiné par OR avec le sprite 5 si ENSP14 a été positionné ;
- les bitplanes impairs, parmi ceux dont les bits ENBP ont été positionnés, et pour le cas où un bit de valeur MVBP a été rencontré.

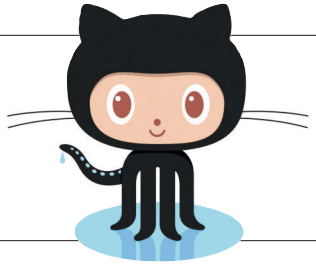
Dans le programme `spritesCollision.s`, les bits 9, 1 et 2 sont ainsi testés.

En permettant de spécifier sur quelle valeur de bit une collision doit être détectée entre un sprite et certains bitplanes, le hardware permet très facilement de spécifier qu'une collision ne doit être détectée que si un sprite rencontre un pixel d'une certaine couleur (éventuellement transparente) dans le décor. Puissant.

Toutefois, cette détection au pixel près n'est pas nécessairement intéressante. Un codeur de jeu vidéo sur Amiga m'a expliqué qu'il convenait mieux d'utiliser des zones de collisions sommaires, incluses dans les entités et le décor, pour moins frustrer le joueur. A cet égard, la détection de collisions du hardware reste rudimentaire. Peut-être a-t-elle été intégrée en référence à une époque où les pixels étaient si gros que cela ne posait pas de problème de gameplay.

La suite de l'article le mois prochain

Retrouvez tous les codes sources des articles sur la page du numéro : **www.programmez.com** et sur notre GitHub : **https://github.com/francoistonic**



Ce n'est pas une app

Une publication Nefer-IT, 57 rue de Gisors, 95300 Pontoise - redaction@programmez.com
Tél. : 09 86 73 61 08 - Directeur de la publication & Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Ont collaboré à ce numéro : S. Saurel, A. Pavie
Nos experts techniques : D. Duplan, J. G. Perrin, C. de Vandiere, D. Anh Pham, D. Van Robaeys, C. Pichaud, P. Charrière, J-B Cazaux, D. Séguy, J. Nourry, F. Jadouani, R. Pacalet, E. Baudoux, Y. Phelizot, J. Mouzin, A. Clerbois. S. Vidouse, S. Graziano, Y. Dehaese, T. Leriche
Couverture : © RapidEye - **Maquette** : Pierre Sandré.
Publicité : François Tonic / Nefer-IT - Tél. : 09 86 73 61 08 - ftonic@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
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, septembre 2018

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.

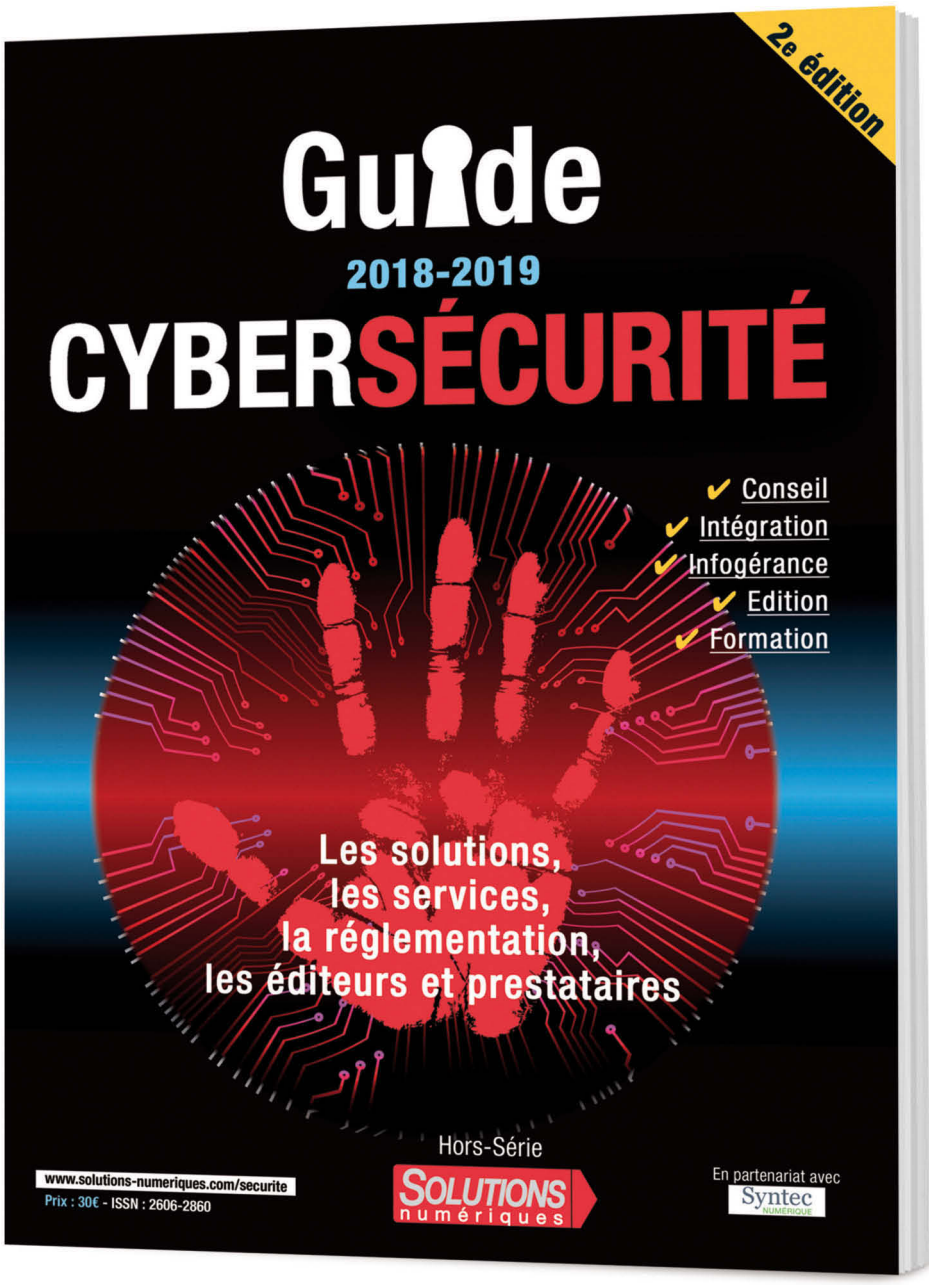
PDF
35 € (monde entier) souscription sur www.programmez.com

2e édition

Suivez le Guide

Le 1er annuaire de la Cybersécurité

Le lexique de la cybersécurité, le RGPD et toute la réglementation,
les acteurs du marché.
Comprendre les solutions et les services de la cybersécurité



Parution
10 octobre 2018

Professionnels : trouvez les informations de référence pour la Cybersécurité
de votre entreprise, ainsi que la présentation des acteurs du marché et des organismes.

Consultez le Guide en ligne :
www.solutions-numeriques.com/securite/

APPLICATIONS CROSS-PLATEFORMES

POUR LES DÉVELOPPEMENTS :

UNE ÉQUIPE WINDOWS ?

+ UNE ÉQUIPE ANDROID ?

+ UNE ÉQUIPE IOS ?

+ UNE ÉQUIPE WEB ?

?

AVEC WINDEV UNE SEULE ÉQUIPE DE DÉVELOPPEMENT SUFFIT !

Avec WINDEV, développez «une seule fois», et créez :

Des applications natives : • Windows • Linux • Mac • Java
+ Des sites pour : • Windows • Linux • ou en PHP
+ Des applications mobiles natives pour smartphones, tablettes et terminaux industriels : • Android, • iOS • UWP • Windows CE.

Elu **«Langage le plus productif du marché»**

Tout est natif.

WINDEV®

VERSION EXPRESS GRATUITE
Téléchargez-la !

WINDEV VU À LA TV
SUR **TF1, M6, BFM TV, CNEWS, LCI**

Tél : 04 67 032 032 info@pcsoft.fr
WWW.PCSOFT.FR