PROGRAMMEZ!
Le magazine des développeurs
N°248
09/10
2021

“Que le code soit avec toi”

Inclus dans ce numéro

- Flutter
- Visual Studio Code + Java
- Deep Learning
- Kotlin
- Ecran e-ink
- Low code
- IoT
- Node
- Angular

Le seul magazine écrit par et pour les développeurs

M 04319 - 248 - F: 6,99 € - RD

Printed in EU - Imprimé en UE - BELGIQUE 7,50 € - Canada 10,55 $ CAN - SUISSE 14,10 FS - DOM Surf 8,10 € - TOM 1100 XPF - MAROC 59 DH

L'intérêt de développer 10 fois plus vite, c'est que vos idées sont plus rapidement sur le marché

VITESSE : UN «TIME TO MARKET» SANS ÉQUIVALENT, QUI VOUS PERMET DE DÉPLOYER EN UN TEMPS RECORD

RECONNAISSANCE : DES CENTAINES DE TÉMOIGNAGES DE CLIENTS SUR PCSOFT.FR (EN TEXTE ET EN VIDÉO)

TOUT EN FRANÇAIS : LOGICIEL, AIDE ET EXEMPLES EN FRANÇAIS (VERSIONS ANGLAISE ET ESPAGNOLE DISPONIBLES)

Cycle de vie • Code multi-plateformes • **Environnement ALM complet** • **Toutes les bases de données** • RAD • Intégration continue • Tableau de bord • **UI/UX: Générateur de fenêtres** • Héritage et surcharge • 70 Contrôles avancés : saisie, planning, bureautique, table, graphe, **cube**, PDF, HTML, carte, treemap, organigramme, champs métier... • Export Excel • Export Word • Sécurité d'accès • **Générateur de rapports** • Codes-barres • .NET • Editeur d'images • **Langage L5G** • **POO** • Editeur de code • Débogueur • Webservices XML, JSON... • Audits • **HFSQL** • Réplication • Cluster • **Versioning** • Tous les standards • 200 Exemples et assistants • Domotique • IOT • RGPD • Tests automatisés • Dossier technique • Télémétrie • Générateur d'aide • Traduction • **Déploiement automatique et maintenance** • Robot de surveillance...

WINDEV
AGL DevOps N°1 en France
Tél: 04.67.032.032

PCSOFT.FR

Document non contractuel. Logiciel professionnel. Time to market = délai de mise sur le marché. WINDEV développe pour Windows et Linux. WEBDEV pour Internet et SaaS. WINDEV Mobile pour iOS, Android et CE

Contenus

Divers

Programmez! est une publication bimestrielle de Nefer-IT.
Adresse : 57, rue de Gisors 95300 Pontoise – France. Pour nous contacter : redaction@programmez.com

SAISON 23 ÉPISODE 248 (OU 252)

« Le calme avant la tempête »

© Excelsior, 17 février 2021. Tournage du Project X

Jeudi 19 août, 14h04 et quelques secondes, pour la première fois, le soleil illumine le ciel. Fini la pluie. Fini les nuages. Que d'enthousiasme, mais sachons garder notre calme. Il est temps de builder ce numéro de Programmez!.

Pas de rachat à 50 milliards, Apple qui explose les chiffres trimestriels, quelques bêta ici et là de Windows et de .Net, quelques nouveautés dans les outils et les JDK. L'été fut finalement relativement calme. Une des grosses annonces de l'été fut la disponibilité de Quarkus 2.0 qui apporte de belles évolutions notamment avec la CLI et le support de la JDK 11. On a aussi pu commencer à tâter Visual Studio 2022 et des nouvelles pré-versions de PHP 8.1.

La fin de l'année s'annonce chaude, très très chaude. Les développeurs auront beaucoup de travail jusqu'à janvier 2022. En vrac :

- Java 17
- .Net 6
- PHP 8.1
- Python 3.10

Si vous n'avez pas encore regardé ces nouvelles versions, nous vous conseillons de commencer à le faire. Si toutes ces versions ne bouleversent pas nos quotidiens, certaines promettent tout de même d'importants changements. .Net 6 est sans doute l'évolution la plus importante avec une refonte des fondations, et pas uniquement de changements de sémantiques. À noter que Microsoft commence à faire le ménage dans le support des versions de Visual Studio 16.x. N'oubliez pas de le mettre à jour pour éviter d'utiliser une version obsolète.

Windows 11 devrait sortir à la fin de l'année, mais une partie des PC ne pourra en bénéficier que l'an prochain. L'interface utilisateur a été dépoussiérée (un peu aux sauces macOS / iOS / Android) et on pourra exécuter des apps Android directement sur Windows, comme on peut déjà le faire avec Linux via le sous-système Linux. Le store subit lui aussi une refonte. Mais rassurez-vous, l'écran de la mort reste bel et bien d'actualité. Ouf ! Par contre, l'OS promet aussi une belle pagaille avec les procédures de sécurité Secure Boot et TPM. Ce dernier composant fait débat. Cette puce obligatoire pour installer Win11 est absente de nombreux PC ayant quelques années. Soit on reste sur un vieux Windows, soit on passe à Linux, soit on change de PC…

En juillet, nous avons eu le plaisir de tester la petite app iDOS. Cette app permettait d'installer MS-DOS et Windows 3.1 sur son terminal iOS. Le résultat était bluffant. Jouer à Prince of Persia ou encore redécouvrir VB 3, le kiff total !

Malheureusement, en août dernier, Apple a retiré l'app du Store pour violation des règles du store. Dura lex sed lex mais j'ai envie de dire : argh ! Pour se consoler un peu, je teste la librairie FabGL. Développée par un Italien, il s'agit d'une librairie graphique native aux ESP32, un MCU surpuissant. L'un des intérêts de la lib est de pouvoir émuler un IBM PC avec MS-DOS et Windows 3.1 ! On peut même utiliser CP/M ou encore émuler un Commodore VIC-20 et même Altair 8800. Pour ce faire, il faut disposer d'un ESP32 équipé d'un port VGA et des connecteurs PS/2 ! Nous vous présenterons ce projet dans le prochain numéro.

Bon 40e anniversaire à deux micro-ordinateurs mythiques :

- IBM PC modèle 5150 : le tout premier PC de l'Histoire
- Commodore VIC-20 : le premier ordinateur à se vendre à plus d'un million d'exemplaires

Bonne rentrée avec Programmez!

François Tonic
Rédacteur en chef depuis longtemps

LES PROCHAINS NUMÉROS

HORS SÉRIE #5 AUTOMNE

100 % technologies Red Hat

Disponible le 18 novembre 2021

PROGRAMMEZ! N°249

.Net 6 / Java 17 / Programmation quantique

Disponible le 29 octobre 2021

Sale été pour la sécurité Windows

L'été n'a pas été clément pour Windows sur le front de la sécurité. Tout a commencé avec la découverte au mois de juin d'une grave faille affectant le spooler d'impression, un composant système permettant de gérer les imprimantes. La faille permettait notamment de prendre le contrôle des appareils vulnérables. Malheureusement, le spooler est présent sur de nombreuses versions de Windows. Les tentatives de l'éditeur pour corriger la vulnérabilité se sont heurtées aux découvertes des chercheurs, qui ont passé l'été à trouver de nouvelles variantes de la faille initiale contournant les correctifs de Microsoft. Parallèlement, de nouvelles failles ont été découvertes dans Exchange mais celles-ci ont eu le bon goût d'avoir été correctement patchées.

Poly Networks, le casse du siècle se finit bien

Poly Networks, une plateforme de finance décentralisée sur la blockchain, s'est réveillé un matin de juillet avec une mauvaise nouvelle à annoncer : la disparition de l'équivalent de 600 millions de dollars en diverse cryptomonnaie, soit le casse du siècle en la matière. Coup de chance pour eux, le coupable (qui a eu le bon goût de rester anonyme) s'est montré conciliant et a progressivement rendu l'intégralité des sommes dérobées. Son geste était selon lui motivé par la volonté de tirer la sonnette d'alarme sur la vulnérabilité ayant permis le vol et s'assurer que celle-ci soit corrigée. Poly Networks s'en tire donc à bon compte, offrant au passage une prime de 500 000 dollars au hacker anonyme. Ce qui montre que pour tirer une prime correcte pour une faille de sécurité, il faut parfois en venir aux grands moyens.

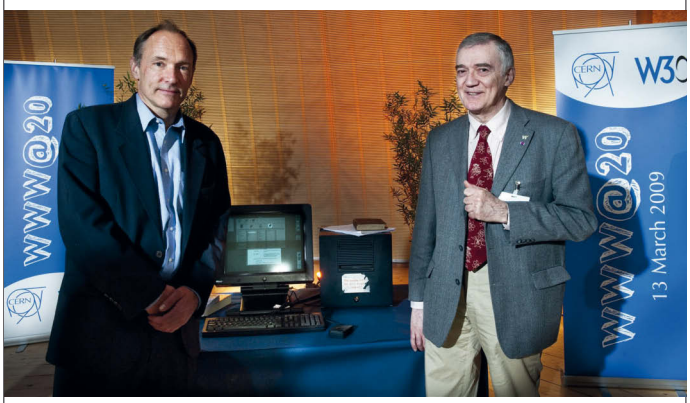
En luttant contre la pédopornographie, Apple ne se fait pas que des amis

Apple a présenté début août son plan visant à mettre en place un système de détection automatisé des contenus pédopornographiques sur les photos et les messages : CSAM, Child Sexual Abuse Material). En soi, cela part d'un bon sentiment : mieux lutter contre les prédateurs sexuels et la pornographie infantile. Mais c'est également un sujet délicat, qui implique qu'Apple, grand défenseur de la vie privée, s'autorise à scanner les fichiers de ses utilisateurs pour y déceler d'éventuels contenus illicites. Largement critiqué pour ce revirement en matière de protection de la vie privée de ses utilisateurs, Apple tente d'expliquer que son système présente des garde-fous et que la société ne pourra pas accéder à l'ensemble des fichiers, uniquement les images identifiées comme illicites.

ARM et Nvidia : les régulateurs traînent des pieds

L'annonce du rachat d'ARM par Nvidia en septembre avait été assortie d'une condition non négligeable : l'accord devait obtenir l'approbation des régulateurs nationaux. Et depuis, les retards se multiplient : Londres indique ne pas voir d'un bon œil le rapprochement entre les deux entreprises, citant des risques liés à la « sécurité nationale », la Commission européenne a fait savoir qu'elle n'avait toujours pas reçu les documents nécessaires à l'ouverture de son enquête et la Chine voit également le deal d'un mauvais œil. Il faudra donc obtenir le feu vert de tout ce beau monde avant de concrétiser le rapprochement et la tâche s'annonce complexe.

Tim Berners Lee cède le code source du web en NFT



Tim Berners-Lee et Robert Cailliau, les deux inventeurs du World Wide Web. La NeXT Station qui a servi à créer le WWW et le Web est exposé au CERN. Photo : CERN

5,4 millions de dollars, adjugé, vendu : c'est le prix auquel Tim Berners Lee a cédé le code source du web. Enfin, il s'agit plutôt d'un « token non fongible » (NFT) certifiant l'authenticité de plusieurs fichiers de code source écrits entre le 3 octobre 1990 et le 24 août 1991 et qui sont présentés comme l'origine du web. Le tout est accompagné d'une lettre signée de la main de son créateur, Tim Berners Lee. Pour le web, cela ne changera pas grand-chose, mais les recettes iront à des œuvres de charité.

Pegasus, le monde du privé facilite l'espionnage

Les révélations du consortium de journalistes Forbidden Stories ont révélé au mois de juillet l'étendue des clients et des cibles du logiciel d'interception légale Pegasus, commercialisé par la société israélienne NSO auprès de services de renseignements et de gouvernement. Sans grande surprise, cet outil initialement destiné à lutter contre des terroristes a également bien servi pour espionner des journalistes, militants, avocats et autres profils jugés gênants par des régimes peu soucieux des droits de l'homme. Le Maroc, bon client de la solution, aurait même ciblé les téléphones de plusieurs membres du gouvernement et d'Emmanuel Macron. Autant dire que cela risque de jeter un froid dans les relations.

Amnesty International, et d'autres organisations dont des membres de l'ONU, plaide pour un moratoire sur la vente des technologies de cybersurveillance.

Bonus de la rédaction

Musk présente son robot humanoïde : Tesla Bot ! Ce robot est de la taille d'un humain. Il se déplace lentement, maximum 8 km/h. Bot a pour but d'aider ou de remplacer les humains dans les tâches dangereuses ou répétitives.

Rendez-vous le 24 septembre sur Apple TV+ pour les premiers épisodes de la série événement Foundations. Oui, oui, l'une des œuvres les plus mythiques de la S-F : le cycle de fondation d'Asimov !

Septembre

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
		1	2	3	4	5
6	7	8	9	10	11	12
	Meetup Programmez!			API Platform Conference (Lille + virtuel)	Angers GeekFest / Angers	
				JUG Summer Camp / La Rochelle		
13	14	15	16	17	18	19
JFTL (Montrouge)						
	WAX, 100 % / Marseille					
20	21	22	23	24	25	26
			DevCon .Net 6 / Azure / Windows (Epitech Paris)			
27	28	29	30			
	Big Data Paris					
	Serverless Days Paris	Devoxx France (Paris)				

Les événements Programmez!

Meetups Programmez!
7 septembre : double meetup agilité + clojure
5 octobre : sécurité du code avec Checkmarx
2 novembre - 7 décembre

Où : WeWorks, 33 rue Lafayette / Paris
Métros : Notre-Dame de Lorette (l12), Le Peletier (l7)
A partir de 18h30

Informations & inscription : programmez.com

Octobre

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
				1	2	3
				Devoxx France		
4	5	6	7	8	9	10
	Meetup Programmez !		Paris Web (Paris)			
			Cloud Bord			
11	12	13	14	15	16	17
18	19	20	21	22	23	24
			DevFest Nantes			
25	26	27	28	29	30	31

Merci à Aurélie Vache pour la liste 2021, consultable sur son GitHub : https://github.com/scraly/developers-conferences-agenda/blob/master/README.md

Les partenaires 2021 de

PROGRAMMEZ!

Le magazine des développeurs

Niveau maître Jedi

softfluent

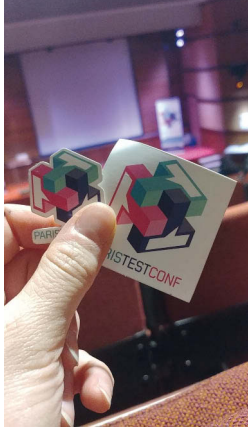
LA MANUFACTURE CACD2

Niveau padawan

OSAXiS

Vous voulez soutenir activement Programmez! ?
Devenir partenaires de nos dossiers en ligne et de nos événements ?

Contactez-nous dès maintenant :
ftonic@programmez.com



Paris Test Conf 3e édition : du 22 au 25 novembre !

Tout a démarré en 2018 sur le slack de La Test Communauté avec quelques passionnés du test logiciel qui ne trouvaient pas dans le paysage français de conférences leur permettant de progresser et surtout de se projeter dans leur quotidien. C'est de là que l'idée de créer leur propre conférence a émergé. Au-delà du contenu même, les organisateurs de la première édition se sont donné plusieurs valeurs que l'événement se doit de porter. La Paris Test Conf, c'est :

- Valoriser le métier de testeur au travers de la notion d'artisan testeur
- Favoriser l'inclusion, la diversité et maîtriser l'impact écologique
- Promouvoir le partage sans but lucratif

La conférence des artisans testeurs

Les organisateurs ont à cœur de promouvoir des conférences qui parlent à l'ensemble des intervenants sur un projet logiciel, du chef de projet aux ops, en passant par les représentants produits, les managers, les business analysts, les développeurs et bien sûr les testeurs. C'est ce qui se cache derrière le terme artisan testeur ! Toute personne contribuante directement ou indirectement à la production de logiciels de qualité est un artisan testeur.

Jean-Pierre Lambert nous parlant de sa vie de testeur

« Lors des deux premières éditions, j'étais très fier des retours faits de proches qui sont sortis grandis de l'événement. Ils ont appris des connaissances leur permettant d'améliorer leurs process en entreprise. » Mimoun K.

L'importance de l'inclusion, de la diversité et de l'impact écologique

Les organisateurs essaient de favoriser la diversité dans les thématiques testing tout en étant soucieux de la représentativité des femmes dans le milieu du test. Lors de la première édition, les organisateurs ont également été sensibles à l'idée de proposer des places gratuites aux personnes en reconversion dans le test et un tarif spécial pour les chômeurs. L'inclusion passe aussi par l'accès aux coulisses de l'organisation à des personnes ne faisant pas partie de l'équipe d'organisation officielle. Sur l'édition en ligne de 2020, les organisateurs ont proposé à des testeurs actifs de la communauté francophone d'héberger et d'animer les soirées. Ces mêmes personnes font aujourd'hui partie de l'organisation de cette troisième édition.

« Quand on a commencé à parler de créer une nouvelle conférence de test en France, je trouvais le projet un peu fou et je ne savais même pas s'il aboutirait. Une version en présentiel et une version en ligne plus tard, je suis très fière de ce que nous avons accompli tous ensemble. Hâte de voir cette troisième édition ! » Lucie D.

Organisateurs de la première édition

Le partage au-delà des bénéfices

Les organisateurs ont toujours souhaité que la ParisTestConf se fasse sans aucun sponsor afin de garder une liberté totale et ne pas faire de compromis. Et pourtant, qu'elle soit en ligne ou présentielle, une conférence a un coût. Lors de la 1ere édition, l'association Agile France a aidé au financement et le prix de l'entrée a permis d'amortir les dépenses (location de la salle, traiteur, etc.). Les bénéfices ont servi à financer d'autres projets associatifs soutenus par Agile France. Les autres éditions, toutes en ligne pour raisons sanitaires, sont gratuites pour les participants. Les organisateurs n'ont pas seulement donné de leur temps mais se sont aussi investis financièrement pour disposer des bons outils et ainsi permettre d'assurer le bon déroulement : site web, billetterie, communication vidéo, etc.

« J'ai été contacté par une organisatrice à la suite d'une intervention dans une conférence sur l'e-commerce. Elle m'a suggéré de faire mon talk à la PTC, je l'ai fait, j'ai été retenu, j'ai bien aimé la conférence, et j'ai ensuite demandé à rejoindre l'organisation. » Simon G.

Des speakers reconnus comme Jean-Pierre Lambert (France) ou Lisa Crispin (USA) et d'autres speakers sont venus partager leur expérience et la passion de la qualité.

Place à l'édition 2021

L'édition 2021 arrive dans quelques semaines et le programme est en cours de préparation. Il devrait être diffusé d'ici la fin de l'été et les inscriptions à l'événement seront ouvertes dans la foulée. Nous espérons vous voir nombreux à cette édition, car une conférence, c'est des organisateurs, des speakers mais avant tout un lieu de rencontre et partage et cela ne sera une réussite que grâce à votre présence !

Les contacts

Twitter @ParisTestConf
LinkedIn : https://www.linkedin.com/groups/8807072
Site : https://paristestconf.com/
Replay des conférences des années précédentes :
https://youtube.com/channel/UCiphrlVAdAc9qt5td1KSovg

Retour d'expérience au travers de l'application Easivio réalisée avec Flutter

Jeremy Favier

Stephane Cadot

Co-fondateurs de Easivio

Le projet Easivio prend naissance fin 2018 avec 2 passionnés provenant du monde du service informatique et ayant essentiellement travaillé sur des projets orientés Web. Nous avions l'idée de proposer une application qui permettrait de diffuser, sur une TV, tout un ensemble de « widgets » configurés au préalable via une application mobile.

Nous voulions que notre application puisse être installée sur iOS et Android. N'étant pas des développeurs mobiles, il était exclu pour nous de nous lancer dans un développement Kotlin (pour la partie Android) et un autre Swift (pour la partie iOS). Nous souhaitions utiliser la même base code pour les 2 plateformes.

De ce constat, il semblait évident que notre choix de framework s'orienterait vers un React Native ou Ionic. Mais Flutter, encore tout récent, semblait tout aussi prometteur.

Ce jeune framework promettait de pouvoir de réaliser des applications cross-platform avec d'excellentes performances. Contrairement aux deux autres frameworks, l'inconnue était le langage de développement : Dart. Mais après la réalisation d'un PoC, il apparut évident que la prise en main de Flutter (et de Dart) était des plus simples. Nous avons pu constater qu'il était assez facile de développer des interfaces plus ou moins complexes tout en ayant une base code compréhensible et maintenable en ayant des performances intéressantes.

Avant de valider Flutter pour notre projet, il nous restait à vérifier 2 points :

- Le support de la communauté
- L'éditeur du framework

Pour un framework aussi jeune, la communauté était déjà bien présente et nous pouvions trouver suffisamment de support en cas de besoin. Quant à l'éditeur du framework qui est Google, il semblait investir suffisamment pour continuer à faire évoluer la plateforme pour en faire le leader du développement d'application mobile cross-platform. Nous ne regrettons pas ce choix. Flutter nous a permis de développer une application complexe, performante, maintenable fonctionnant sur iOS et Android depuis la même base code.

Architecture globale

L'architecture globale du projet est relativement simple (cf. **Figure 1** - Architecture globale).

Le projet repose sur 2 applications développées entièrement avec Flutter :

- Une application servant à configurer les widgets de l'utilisateur : Easivio
  - Cette application est installée sur un smart phone ou une tablette
- Une application permettant d'afficher ses widgets : Easivio TV
  - Celle-ci est à destination des SmartTV, Box ou même tablette tant que le système d'exploitation reste une base Android ou iOS

L'application Easivio TV se connecte au travers d'API externes afin de récupérer les données à afficher sur les widgets.

Chaque application s'appuie sur Firebase pour stocker et récupérer les informations liées notamment à l'affichage des widgets. Dans le cadre de ce projet, le choix de Firebase était presque une évidence. Firebase est une plateforme dédiée à la création d'application mobile et web, elle offre un ensemble d'outils permettant de :

- Construire son application (Gestion de l'authentification, BD NoSQL, Stockage de fichier, hébergement d'application web, déploiement de cloud function…)
- Monitorer son application
- Analyser l'utilisation de son application
- Et bien d'autres encore…

Au-delà de tous ces outils facilitant la création d'applications mobile et web, son intégration dans une application Flutter est très simple. Il existe un package Flutter pour chaque outil. Par exemple :

- Pour la gestion de l'authentification avec Firebase : firebase_auth
- Pour la connexion à Firestore : cloud_firestore
- Pour Crashlytics : firebase_crashlytics
- …

L'ensemble des packages sont disponibles sur https://pub.dev.

Prenons le cas d'une nouvelle application qui souhaite utiliser Firestore pour y persister l'ensemble de ses données. Il faudra tout d'abord créer son projet dans Firebase afin de pouvoir déposer le fichier google-services.json dans le projet Flutter. C'est grâce aux informations contenues dans ce fichier qu'il est possible d'établir une connexion entre l'application mobile et Firebase.

Ensuite, il faudra importer le package cloud_firestore au projet. Pour cela, il suffit d'éditer le fichier pubspec.yaml (qui

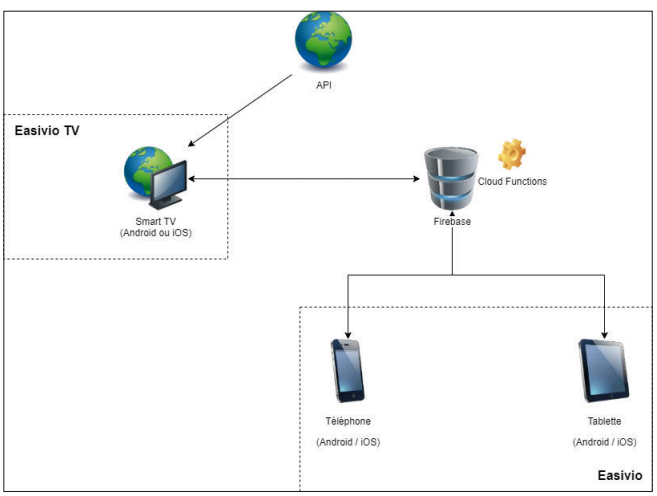


Figure 1
Architecture globale

recense toutes les dépendances du projet) et d'y insérer le package comme montré à la **Figure 2**.
Enfin, importez le package dans votre classe Dart et coder vos différentes requêtes de lecture ou écriture. Par exemple, la **Figure 3 -** permet de récupérer un document dans la collection des utilisateurs depuis son identifiant..

```
import 'package:cloud_firestore/cloud_firestore.dart';

Future<UserModel> findUserBy(String uuid) async {
 final DocumentSnapshot ds = await FirebaseFirestore.instance
    .collection(collectionDBName)
    .doc(uuid)
    .get();
 return ds.exists
    ? UserModel.fromJson(ds.data()!)
    : throw UserNotFoundException();
}
```

Figure 3 - *Firebase rechercher un utilisateur.*

Firestore offre également la possibilité « d'écouter » les changements d'un (ou plusieurs) document et de les notifier dès lors. Le code de la **Figure 4** - permet d'être notifié dès qu'un changement survient sur le document Firestore identifié par l'uuid passé en paramètre dans la collection **collectionDBName**.

```
void listenOnUserChange(String uuid) async {
 FirebaseFirestore.instance
    .collection(collectionDBName)
    .doc(uuid)
    .listen((DocumentSnapshot ds) {
  // DO SOMETHING
 });
}
```

Figure 4 - *Listen sur un user*

Voici quelques exemples montrant la facilité d'intégration et d'utilisation d'une plateforme telle que Firebase dans un projet Flutter.

La communauté Flutter

Au cours de notre projet, nous avons trouvé sur différents canaux d'échanges et d'entraide les solutions aux problèmes techniques que l'on a pu rencontrer.
On peut s'apercevoir de l'engouement autour de Flutter avec le nombre de packages dédiés à Flutter qui ont été publiés sur pub.dev. Aujourd'hui, on en retrouve plus de 15K. Néanmoins, il faut faire attention dans le choix des packages que l'on utilise au sein de son projet.
Intégrer un package externe dans son projet permet d'accélérer son développement sans avoir à recréer quelque chose qui existe déjà. Mais il faut garder à l'esprit que cela crée une dépendance entre votre projet et ce package. Lors du choix d'un package, il est important de s'assurer de la pérennité de celui. Vous ne pourrez jamais être sur à 100% qu'un package sera maintenu mais certains indices vous permettent d'écarter ceux qui paraissent obsolètes :

- La date de dernière mise à jour
  - un package qui n'a pas été mis à jour depuis un an ou deux n'est sûrement plus maintenu.
- L'activité sur le Github du package
  - un package peut avoir des anomalies mais il est important de s'assurer que les mainteneurs du package répondent aux anomalies levées et soient réactifs. Si aucune activité n'apparait sur le github du package, celui-ci n'est probablement plus maintenu
- Le package est identifié comme faisant partie du programme Flutter Favorite. Il s'agit des packages que l'équipe Flutter considère comme étant à privilégier pour le développement d'une application Flutter (https://flutter.dev/docs/development/packages-and-plugins/favorites). Il est rare que ces packages deviennent obsolètes.
- Le package est taggué **Null Safety**. Depuis quelques mois Flutter supporte le null safety. En cas de démarrage d'un nouveau projet, ce dernier sera, lui aussi NullSafety, compatible. Beaucoup de packages ont été développés avant le support du nullsafety et ont dû migrer afin d'assurer la compatibilité avec Flutter 2+. Néanmoins, certains packages n'ont pas été migrés. Il est fort probable que ces packages ne soient plus maintenus.
- Le nombre de **likes** d'un package sur pub.dev peut être un indicateur quant à la pérennité d'un package. En effet, plus un package est liké et utilisé, plus il y a de chance que celui-ci soit maintenu du fait de sa popularité.

Restez vigilant aux dépendances que vous créez avec votre projet. Certains packages peuvent s'avérer difficiles à remplacer s'ils deviennent obsolètes et sont structurants pour votre projet.

Provider

Il faudrait plusieurs articles pour couvrir la totalité des fonctionnalités que propose Provider. Son auteur (Rémi Rousselet) décrit son package comme étant un mélange entre de la gestion d'état et de l'injection de dépendance. D'autres décrivent Provider comme un package permettant aux « Humains d'utiliser les InheritedWidget ».
Les InheritedWidget, inclus dans le SDK Flutter, servent à propager des informations au travers de votre arbre de widgets. Leur utilisation est complexe et pas vraiment intuitive, c'est l'une des raisons pour lesquelles Provider a été développé ; il va faciliter le mécanisme de propagation des informations dans votre application.
Dans le cadre du développement d'Easivio, nous avons utilisé Provider de deux manières différentes :

- En tant qu'injecteur de dépendance pour nos services métiers
- En tant que gestionnaire d'état

La documentation d'intro Flutter sur ce sujet est bien faite et permet de bien comprendre les enjeux d'un gestionnaire d'état (https://flutter.dev/docs/development/data-and-backend/state-mgmt)

Provider – Injection de dépendance

Dans Easivio, nous avions besoin de pouvoir accéder à la même instance de certains services métiers depuis divers endroits de l'application. Nous avons utilisé Provider pour appliquer le principe d'injection de dépendance de nos services.

```
1  dependencies:
2    cloud_firestore: ^1.0.7
```

Figure 2 - *Dépendance vers Firestore*

Pour cela, il suffit de déclarer un MultiProvider au-dessus de votre MaterialApp (ou du Widget qui sera le plus haut dans votre arbre accédant à vos services). Ce MultProvider permettra d'instancier tous les services qui seront accessibles depuis le reste de l'application. Dans l'exemple **Figure 5** - Les services Service1, Service2 et Service3 pourront être utilisés par l'ensemble des Widget fils de la MaterialApp.

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

class MyApp extends StatelessWidget {
 const MyApp({Key? key}) : super(key: key);

 // This widget is the root of your application
 @override
 Widget build(BuildContext context) {
  return MultiProvider(
   providers: [
    Provider < Service1(create: (_) => Service1()),
    Provider < Service2(create: (_) => Service2()),
    Provider < Service3(create: (_) => Service3()),
   ],
   child: MaterialApp(
    title: 'Flutter demo',
    theme: ThemeData(primarySwatch: Colors.blue),
    home: MyHomePage(title: 'Flutter Demo Home Page'),
   ),
  );
 }
}
```

Figure 5 - *Provider - Injection de dépendance*

Pour récupérer l'instance de l'un des services depuis un descendant de la MaterialApp, il suffit d'utiliser l'une des deux nomenclatures de la **Figure 6** :

```
final Service1 service1 = Provider.of<Service1>(context, listen: false);
final Service1 service1 = context.read<Service1>();
```

Figure 6 - *Provider - Récupérer un service*

Provider – Gestionnaire d'état

L'interface graphique est fortement liée aux différents états que l'on retrouve dans l'application. Un même état peut avoir un impact sur plusieurs Widgets de l'arbre Flutter.

Prenons l'exemple de la **Figure 7** - où la **Value** du widget de plus haut niveau correspond à l'état de cet arbre. Cette **Value** est également utilisée par deux widgets feuilles de l'arbre. A chaque changement d'état de **Valeur**, il est donc nécessaire de reconstruire tous les widgets de l'arbre se trouvant entre le widget de plus haut niveau et les widgets ayant besoin de **Value** afin de passer **Value** en paramètre. Il s'agit là d'un comportement assez verbeux et non performant.

L'utilisation de Provider (s'appuyant sur les InheritedWidget) permet de nettement simplifier et optimiser ce comportement. L'état **Value** est déplacé dans un Provider et des widgets Consumer sont insérés au-dessus de chaque Widget ayant besoin de l'état. Ces Consumers écoutent les changements survenant sur **Value** et reconstruire uniquement leurs descendants lorsqu'un changement survient. Ce principe permet de :

- Faciliter l'écriture du code

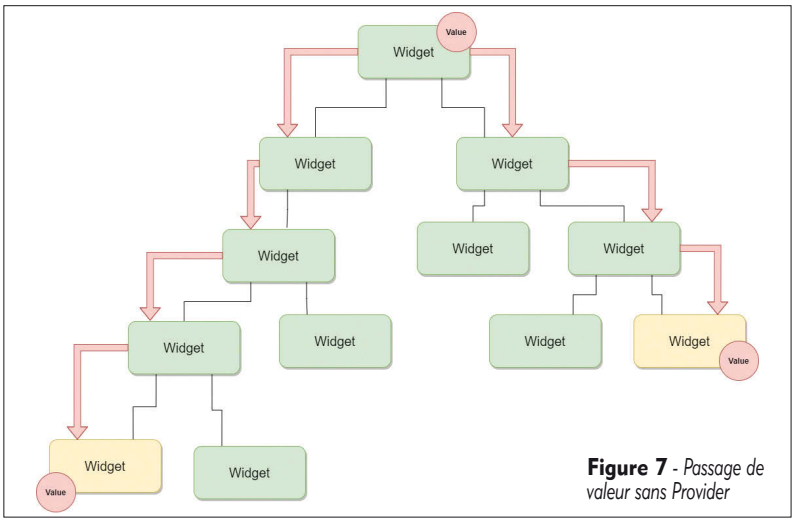


Figure 7 - *Passage de valeur sans Provider*

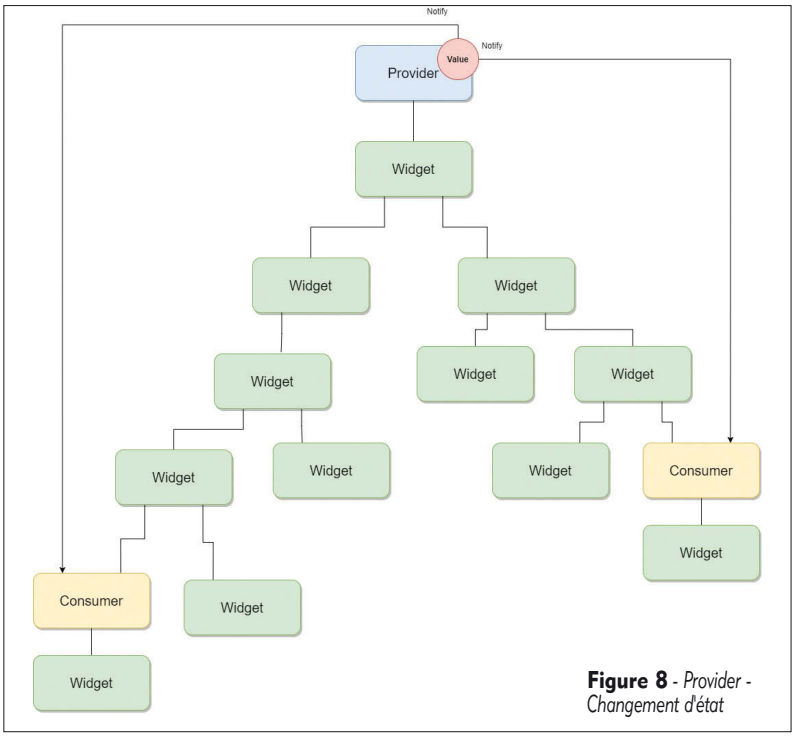


Figure 8 - *Provider - Changement d'état*

  - Il n'est plus nécessaire de passer les valeurs en paramètre aux widgets
- Limiter et maitriser les builds dans l'arbre des widgets
- Sortir l'état de l'application (donc une partie de la logique métier) de nos Widgets
  - Un point que nous reverrons dans la partie Architecture interne

Figure 8

Architecture interne

Il est important de réfléchir au choix de l'architecture interne d'une application avant de commencer son développement. Il faut garder en tête que votre application doit être :

- Maintenable
- Testable
- Évolutive

La mise en place d'une mauvaise architecture (ou même d'aucune réelle architecture ou réelle organisation) pourrait rendre votre application difficilement maintenable, testable ou évolutive. Dans le cadre de notre projet Easivio, nous avons utilisé le pattern MVVM (Model-View-ViewModel).

Ce pattern permet de déplacer la logique métier dans les couches Model et ViewModel, laissant la vue responsable uniquement du rendu graphique et du déclenchement des actions utilisateurs.

Le Model : Il est le garant des données et de la logique métier. C'est lui qui aura la charge de récupérer (et persister) les données depuis les sources externes (DB, API, …).

Le ViewModel : Il est vu comme un médiateur entre le Model et la View. Il reçoit toutes les demandes des View (ex : tap sur un bouton) et appelle le Model en fonction. Une fois que le Model lui a fourni une réponse, il notifie la View pour l'avertir que les données sont disponibles.

La View : Elle représente l'interface graphique de l'application et permet d'interagir avec l'utilisateur. Elle va capter les évènements générés par l'utilisateur et les transmettre au ViewModel. Elle est également en écoute sur les notifications que le ViewModel peut renvoyer afin de savoir à quel moment elle doit se mettre à jour.

La **Figure 9** montre les interactions entre les différentes couches de l'architecture.

Prenons un exemple simplifié pour illustrer les échanges entre les différentes couches. Nous nous appuierons sur Provider pour la gestion d'état de cet exemple : **Figure 10**

Lors du build de la View, elle crée son ViewModel et se met en écoute sur les changements de la variable ***productPrice*** qui se trouve dans le ViewModel.

Nous utilisons Provider (ligne 30) pour se mettre en écoute sur les changements de **productPrice** et rebuild le widget si la valeur change.

L'utilisateur peut déclencher la méthode ***searchProductPrice*** du ViewModel en appuyant sur le ***FloatingActionButton***.

Le Widget représentant l'UI est le descendant du ChangeNotifierProvider, il s'agit là de l'application en code de la **Figure 8**.

Les méthodes ***context.select()*** et ***context.read()*** font appel au ViewModel via Provider. On note ici le découplage grâce à Provider de la View et de la logique métier.

Figure 10 - View - HomePage :
Code complet sur programmez.com & github

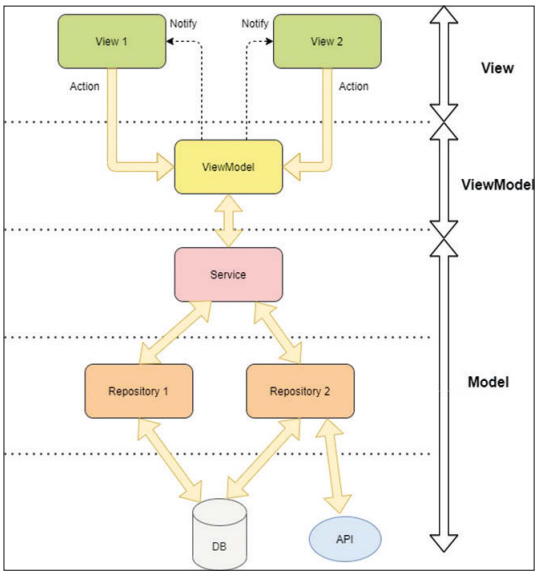


Figure 9 - *Architecture MVVM*

Le ViewModel - home_page_view_model.dart (Figure 11)

La méthode ***searchProductPrice*** du ViewModel se chargera d'appeler le service métier (couche Model) permettant de retrouver le prix du produit. Au retour du service, il met à jour sa variable de classe ***productPrice*** et notifie la View qu'un changement a eu lieu via l'appel de ***notifyListeners()***.

```
import 'package:flutter/material.dart';
import 'package:programmez/services/catalog_service.dart';

class HomePageViewModel with ChangeNotifier {
 HomePageViewModel() {
  _catalogService = CatalogService();
 }

 late final CatalogService _catalogService;
 double productPrice = 0.0;

 Future<void> searchProductPrice(String productId) async {
  productPrice = await _catalogService.searchProductPrice(productId);
  notifyListeners();
 }
}
```

Figure 11 - *ViewModel – HomePageViewModel*

Le Model - catalog_service.dart (Figure 12)

Le CatalogService, quant à lui, récupère les données depuis la couche repository en charge de communiquer avec les systèmes externes. Il applique les règles métiers nécessaires, ici on calcule la TVA du prix du produit via la méthode ***computeVAT()*** et on renvoie le résultat à l'invocateur du service.

```
import 'package:programmez/core/product.dart';
import 'package:programmez/repositories/catalog_repository.dart';

class CatalogService {
 CatalogService() {
  _catalogRepository = CatalogRepository();
 }

 late final CatalogRepository _catalogRepository;

 Future<double> searchProductPrice(String productId) async {
  final Product product = await _catalogRepository.findProduct(productId);
  return computeVAT(product.price);
 }

 double computeVAT(double price) {
  return price * 1.2;
 }
}
```

Figure 12 - *Model – CatalogService*

Le Model – catalog_repository.dart (Figure 13) Le CatalogRepository récupère dans firebase le document correspondant à l'id du produit. Il se charge de désérialiser le document récupéré et renvoie un objet de type ***Product***.

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:programmez/core/exception.dart';
import 'package:programmez/core/product.dart';
```

```
class CatalogRepository {
 Future<Product> findProduct(final String productId) async {
  final DocumentSnapshot ds = await FirebaseFirestore.instance
     .collection('catalog')
     .doc(productId)
     .get();
  return ds.exists
     ? Product.fromJson(ds.data()!)
     : throw ProductNotFoundException();
 }
}
```

Figure 13 - *Model - CatalogRepository*

On s'aperçoit avec ce pattern MVVM qu'il est alors plus aisé d'apporter des modifications aux couches View ou Model sans impacter fortement le reste de l'application. Nous pourrions, ici, choisir de changer de base de données pour passer de Firebase à une autre BD. Pour cela, seul ***catalog_repository.dart*** devrait être modifié. A l'inverse, on pourrait adopter un changement de design au niveau de la View. Dans ce cas, seul ***home_page.dart*** serait changé.
Avec la mise en place de ce pattern sur l'application Easivio, la maintenance quotidienne et les évolutions en sont simplifiées. Au cours du développement de l'application, nous avons dû faire certains refactoring qui se sont vus plus simples que prévu du fait de la mise en place d'une telle architecture. Seule une couche étant impactée, les risques devenaient limités et le temps de mise en œuvre réduit.

Le design

Fin 2019, les 2 tendances qui se démarquaient en termes de design étaient l'utilisation de gradients et le Neumorphisme.

Neu-moph-koi ?

A l'origine, le skeumorphism. Du grec « skéuos » signifiant « contenant ou outil » et de « morph » signifiant « forme », il est utilisé pour exprimer l'utilité de l'élément avec sa représentation réelle. La corbeille papier comme poubelle, la disquette pour la sauvegarde, le bloc note, la boussole sont des références à notre réalité, permettant d'appréhender la fonction aisément. Le swipe est un autre exemple, il imite le fait de tourner la page d'un livre.

Figure 14 - *Exemple de skeumorphism*

Utilisé jusqu'à iOS 6, le flat design le remplace à partir de 2013. Ce design se veut plus moderne et simplifié. A la suite d'Apple, Google l'adapte avec Material Design. Plus récemment, Microsoft s'y met aussi, notamment dans Windows 11.

Une évolution : le neumorphism

Le neumorphism est un compromis :

- Au flat design, il ajoute un effet de profondeur aux objets qu'il représente
- Au skeumorphism, il reprend l'effet permettant de différencier un bouton activé de sa version inactive en créant une fausse 3D.

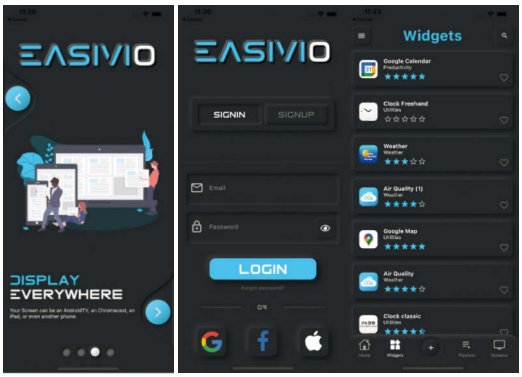


Figure 15 - *Design Easivio*

Dans Easivio, le neumorphism permet donc d'identifier rapidement les boutons, les états tels que la page sélectionnée ou de délimiter des éléments graphiques (TextField, Boutons, Cards)
Les différents effets : **Figure 16**

Comment le mettre en place ?

- Flat : Pour réaliser cet effet, nous partons d'un container, et lui ajoutons 2 ombres opposés. Ces ombres sont le résultat d'une lumière source.

Figure 17 - *Classe de base d'un NeuTestContainer de type Flat :*
Code complet sur programmez.com & github

- Convex : Pour avoir un effet convexe, il suffit d'ajouter un gradient à notre BoxDecoration.

```
decoration: BoxDecoration(
   borderRadius: borderRadius,
   border: boxBorder,
   color: color,
   gradient: _getGradient(color, intensity, neumorphicType, lightSource),
```

Figure 18 - *Ajout d'un gradient à BoxDecoration*

Celui-ci va varier en fonction de l'intensité et de la source lumineuse.

- Concave : On inverse le gradient comme dans le snippet ci-dessous.

```
/// Retourne un [Gradient] d'une liste derivé de [BaseColor]  en fonction du
 [NeumorphicType]
/// prend [lightSource] en compte pour l'orientation du gradient
Gradient? _getGradient(
  Color baseColor, int intensity, NeumorphicType type, Offset lightSource) {
 List<Color>? gradientColors;
 switch (type) {
  case NeumorphicType.concave:
   gradientColors = getColors(baseColor, intensity);
   break;
  case NeumorphicType.convex:
```

Figure 16 - *Neumorphisme sur un Container. De gauche à droite (Flat, Emboss, Convex, Concave)*

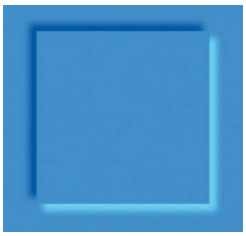


Figure 21 - *Résultat de l'inversion des ombres*

```
    gradientColors = getColors(baseColor, intensity).reversed.toList();
    break;
  case NeumorphicType.emboss:
  case NeumorphicType.flat:
  default:
    return null;
 }

 return LinearGradient(
    begin: Alignment(lightSource.dx, lightSource.dy),
    end: -Alignment(lightSource.dx, lightSource.dy),
    colors: gradientColors);
}
```

Figure 19 - *Récupération du gradient en fonction du type*

- Emboss : A première vue, on reprend notre code du flat mais on change la direction des ombres pour qu'elles soient portées à l'intérieur du container.

```
/// les ombres internes au container ?
final List<BoxShadow> shadowList = <BoxShadow>[
 BoxShadow(
    color: getAdjustedColor(color, intensity),
    offset: -offset, // au lieu de -offset
    blurRadius: blur),
 BoxShadow(
    color: getAdjustedColor(color, 0 - intensity),
    offset: offset, // au lieu de offset
    blurRadius: blur),
];
```

Figure 20 - *Inversion des ombres*

Le résultat n'est pas vraiment celui escompté. Les ombres ont bien été inversées mais elles sont appliquées sous notre container.
Figure 21
Pour peindre sur notre container, on peut créer notre propre RenderObject et surcharger la méthode paint.

Figure 22 - *Classe permettant la création d'ombres sur ses widgets enfants :* ***Code complet sur programmez.com & github***

Et nous revoilà avec notre effet tel que voulu au départ.

```
return neumorphicType != NeumorphicType.emboss
      ? Container(
         height: height,
         width: width,
         decoration: BoxDecoration(
           color: color,
           boxShadow: shadowList,
           gradient:
              _getGradient(color, intensity, neumorphicType, lightSource),
         ),
        )
      : InnerShadow(
          key: key,
          shadows: shadowList.reversed.toList(),
          child: Container(
           height: height,
           width: width,
           decoration: BoxDecoration(
            color: color,
           ),
          ));
```

Figure 23 - *Selection du widget en fonction de l'enum*

En ajoutant un child et les autres propriétés du container, vous obtiendrez ainsi un NeuMorphicContainer.
Comme souvent avec Flutter, il existe plusieurs librairies que vous pouvez intégrer directement à votre projet ou vous en inspirer pour une implémentation plus poussée. flutter_neumorphic (https://pub.dev/packages/flutter_neumorphic) est celle dont nous nous sommes inspirés.

Les impacts

Performances

L'utilisation d'effets doit être effectuée avec précaution comme indiqué dans les bonnes pratiques de Flutter. (https://flutter.dev/docs/perf/rendering/best-practices)
Les ShaderMask, Clipping, et l'Opacité sont des opérations pouvant être coûteuses pour la partie graphique.
Or pour un élément Neumorphique, nous en utilisons jusqu'à 4 (un gradient, un clip pour la forme, et 2 ombres). **Figure 24**
On le constate avec l'outils de performance DevTools, le coût de construction des widgets est faible (Bleu clair) mais l'impact sur le GPU est significatif (Bleu foncé).

Accessibilité

Suivant le contraste du téléphone, l'intensité de l'effet choisi, il peut être difficile pour certaines personnes de bien différencier les éléments. C'est l'une des plus grosses limitations de ce design.

```
border: Border.all(color: Colors.blue[800]!, width: 2),
```

Figure 25 - *Exemple de bordure pour un Container*

Une solution serait d'intégrer une bordure pour intensifier l'effet.

Conclusion

Vous souhaitez développer une application mobile mais vous ne savez pas quel framework ou langage utiliser ? Vous pouvez partir avec Flutter les yeux fermés.
Depuis un peu plus de 2 ans, nous travaillons avec Flutter et nous n'avons pas rencontré de mauvaises surprises. Bien au contraire, ce framework a tenu toutes ses promesses et semble bien continuer à évoluer au fil de ses versions. On a pu voir à la sortie de Flutter 2.0 que la version Web était désormais stable, ce qui permet de créer avec une seule base code (ou presque) une application compatible Android, iOS et Web. Google a également annoncé que Flutter 2.0 était stable pour les plateformes Windows, macOS et Linux. D'ailleurs, les nouvelles applications de Canonical (Ubuntu) seront désormais développées avec Flutter.
Tout cela laisse entrevoir des perspectives immenses en termes de développement d'applications et de compatibilité de plateforme.

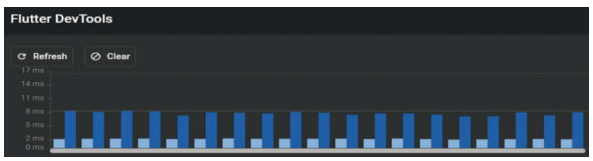


Figure 24 - *Impact sur les performances de la mise en place des effets*


L'HEURE
</DU DEV>
NOUVELLE SAISON À
PARTIR DE SEPTEMBRE
Une émission
100% développeurs
Actualité / Carrière
Compétences / Live coding
Pour découvrir en
avant-première
la nouvelle saison,
inscrivez-vous ici
Replay des émissions de la saison 1
https://www.xsior.com/category/lheure-du-dev/
L'heure du Dev, sponsorisé par IBM,
le Cloud public le plus ouvert
et le plus sécurisé du marché.
IBM
pro·gram·mer
an organism that tur
pizza
47
excelsior.

Grégory Bersegeay
MVP Embarcadero (Delphi), développeur fullstack, conférencier.

Labyrinthe en 3D avec Delphi

Le framework Firemonkey (FMX) de Delphi dispose de fonctionnalités 3D. Sans avoir à installer un moteur particulier, nous pouvons aisément ajouter de la 3D à nos applications. Cerise sur le gâteau, FMX étant multiplateformes, le même code fonctionnera sur desktop et sur mobile ! Nous allons vous montrer toute la puissance du framework et de la 3D avec la programmation d'un jeu 3D.

FMX fournit pour la partie graphique une couche d'abstraction qui s'appuie sur l'API graphique utilisée par le système d'exploitation cible comme indiqué à la **figure 1**.

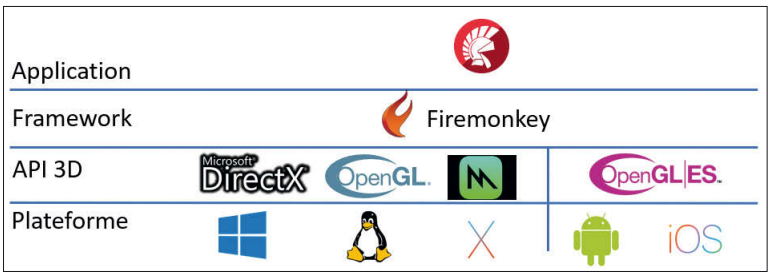


Figure 1

Les versions de Delphi antérieures à la version 10.3 (Rio) s'appuyaient sur OpenGL sous macOS. Depuis la version 10.3, il est possible d'utiliser l'API Metal à la place d'OpenGL. Le support de Metal a été amélioré avec la version 10.4 (Sydney). En plus de suivre les recommandations d'Apple, utiliser Metal plutôt qu'OpenGL permet d'obtenir de meilleures performances.
Pour activer l'utilisation de Metal, il suffit d'ajouter dans le code source du projet Delphi les lignes conditionnelles suivantes :

```
{$IFDEF MACOS}
 GlobalUseMetal := True;
{$ENDIF}
```

Nous ne verrons pas dans cet article l'utilisation des shaders (programmes qui s'exécutent sur le GPU et non sur la CPU). Sachez que c'est possible (d'ailleurs les effets graphiques fournis en standard avec FMX sont des shaders), mais dans ce cas, vous devrez écrire vos shaders dans le langage supporté par l'API graphique utilisée.
Par exemple, si vous écrivez une application multiplateforme en utilisant les shaders, vous devrez écrire les shaders en HLSL (High Level Shader Language) pour Windows (DirectX), en GLSL (GL Shader Language) pour Android, iOS, Linux et macOS (OpenGL) et en MSL (Metal Shader Language) pour macOS (Metal).

Exemple avec un jeu

Les fonctionnalités 3D fournies avec FMX ne rivalisent pas avec ce que peuvent proposer des moteurs tels que Unity ou Unreal. Embarcadero le signale justement en indiquant que ce n'est pas un moteur de jeu. En revanche, ces fonctionnalités 3D peuvent être intéressantes dans le cadre d'applications industrielles, scientifiques ou éducatives.
Toutefois, il est possible de réaliser des jeux en 3D ! À la suite de mon article paru dans Programmez! n°247 sur un exemple de jeu de plateforme en 2D, l'exemple qui servira de fil rouge à cet article sera également un jeu. Cette fois-ci, il s'agira d'un jeu de labyrinthe en 3D où le joueur devra trouver la sortie. Avant de pouvoir sortir du labyrinthe, il faudra activer la sortie en appuyant sur un interrupteur qu'il faudra également trouver.
Le labyrinthe sera généré aléatoirement au lancement d'une partie. La sortie et l'interrupteur seront également placés aléatoirement dans le labyrinthe généré. Ainsi, chaque partie sera différente !
Les liens pour télécharger le code source et les binaires du jeu sont disponibles à la fin de l'article. Le projet est compatible avec les versions de Delphi à partir de la 10.3 Rio, cela signifie qu'il est compatible avec l'édition Community (gratuite sous certaines conditions) : https://www.embarcadero.com/fr/products/delphi/starter

Repère

Avant de débuter le développement en 3D, deux choses sont à connaître : l'orientation du repère et la manière de positionner des objets 3D. Sous FMX, en 3D, le repère est orienté comme indiqué sur la **figure 2**.

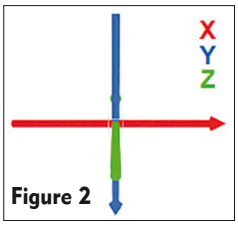


Figure 2

Nous constatons que l'axe des abscisses (en rouge sur la **figure 2**) est orienté vers la droite, l'axe des ordonnées (en bleu sur la **figure 2**) est orienté vers le bas et l'axe des cotes (en vert sur la **figure 2**) est orienté vers le fond de l'écran.
En plus du repère, comprendre le placement des objets dans la scène 3D (position, orientation et taille) est primordial. Sous FMX, la position d'un objet en 3D correspond à la position du centre de l'objet. La **figure 3** illustre cela avec un cube : son centre étant le point d'intersection des diagonales. Lorsqu'on positionne le cube en jouant avec sa propriété **position**, nous jouons sur la position du centre du cube.

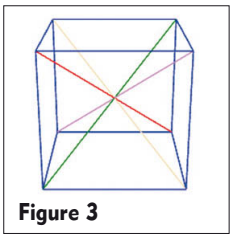


Figure 3

C'est également à partir de ce centre que l'on taille l'objet. Imaginons que nous positionnons le cube à la position x=0, y=0 et z=0. Par défaut, le cube a une largeur, une hauteur et une profondeur de 1. Il occupe alors le volume de x = -0,5 à x = 0,5, de y = -0,5 à y = 0,5 et de z = -0,5 à z = 0,5. La **figure 4** montre un tel cube avec les axes gradués du repère (en bleu).

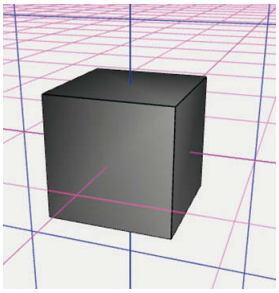


Figure 4

Si nous passons sa largeur à 2 par exemple, il deviendra un rectangle en 3D et occupera le volume de x=-1 à x=1, de y=-0,5 à y=0,5 et de z=-0,5 à z=0,5 (comme illustré à la **figure 5**).

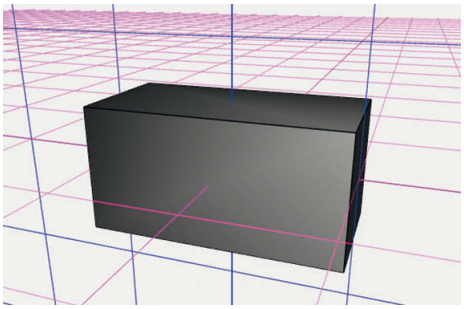


Figure 5

Lorsqu'on dimensionne un objet, il faut bien avoir à l'esprit que ce dimensionnement se fera à partir du centre de l'objet et dans les deux sens de l'axe concerné.

Comme toujours avec Delphi, on peut créer, placer, dimensionner et orienter les composants via le code bien sûr, mais également via l'éditeur graphique. C'est valable pour les éléments 2D (listes, cases à cocher, zones d'édition, boutons…), mais également en 3D. La scène 3D peut alors être composée à la souris et on visualise en temps réel le rendu. Dans l'IDE, le rendu est purement logiciel, mais à l'exécution, le programme utilisera par défaut le rendu matériel si votre GPU le permet.

Le rendu matériel utilisera votre GPU (processeur graphique). Plus celui-ci sera puissant, meilleures seront les performances de l'application 3D. Pour son activation, FMX réclame au minimum un GPU compatible :

- Pour Windows : DirectX 11 (meilleur avec DirectX 11.1, car utilisation du Pixel Shader Level 5) ;
- Pour macOS : si votre mac est certifié au moins pour macOS 10.12 Sierra ou plus, votre GPU sera compatible;
- iOS 10 ;
- Android : un processeur au minimum ARMv7 avec le support de NEON
- Linux : OpenGL à partir de la version 3.3 (meilleur à partir de la version 4.3, car utilisation du shader language 4.3)

Si ce n'est pas le cas de votre configuration, le rendu logiciel sera alors utilisé, mais l'application sera bien moins performante.

La **figure 6** montre le bureau Delphi en mode conception 3D. Il est possible d'interagir avec les composants 3D avec la souris. Dans cette capture d'écran, l'objet Cube1 est sélectionné (nous voyons ses propriétés dans l'inspecteur d'objet) et les petites poignées bleues visibles dans la conception graphique permettent de tailler et d'orienter le cube à la souris. Le cube jaune représente un objet TLight (la source de lumière de la scène exemple, ici en mode spot) qu'il est possible

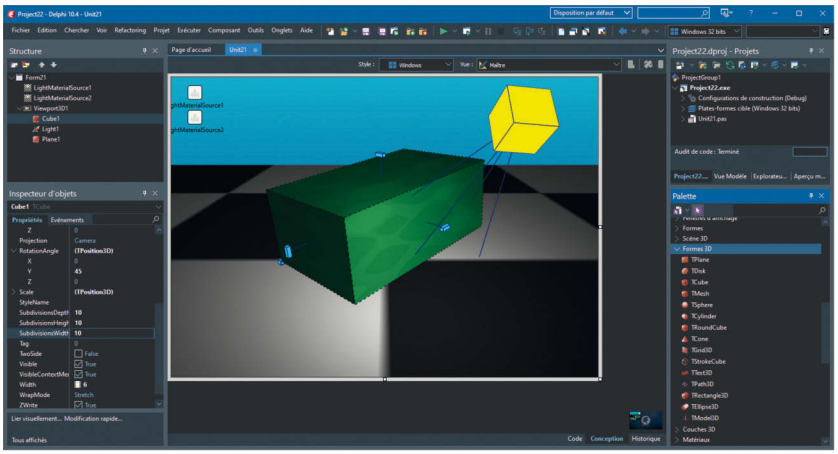


Figure 6

évidemment de déplacer et d'orienter différemment à l'aide de la souris. Le résultat est affiché en temps réel.

Création du projet sous Delphi

Il existe deux manières de débuter un projet 3D sous Delphi avec Firemonkey. Soit vous créez un nouveau projet à partir de l'assistant « Application 3D » soit en partant de l'assistant « Application vide ». Le premier vous crée une fiche qui intègre déjà un contexte 3D, alors que le second va nécessiter d'ajouter un composant **TViewport3D** sur la fiche créée par défaut.

Il n'y a ensuite pas de différence particulière. Le fait d'utiliser le composant **TViewport3D** permet d'ajouter simplement et facilement une vue 3D à une application 2D existante par exemple.

Dans le jeu que nous allons détailler dans cet article, je suis parti d'un projet vierge auquel j'ai ajouté un composant **TViewport3D**.

Vous trouverez dans le code source du projet 3 unités : uMain.pas correspond à l'unité principale et elle contient l'interface graphique, uLabyrinthe.pas contient les méthodes de génération du labyrinthe et uUtils3D.pas contient des méthodes utilitaires 3D (en particulier pour la gestion des collisions que nous verrons plus tard).

Génération d'un labyrinthe

Notre projet étant un jeu où le joueur devra activer et trouver la sortie d'un labyrinthe en 3D, le premier élément à générer sera le labyrinthe. Celui-ci sera généré aléatoirement afin de rendre chaque partie unique et ainsi d'augmenter la durée de vie du jeu.

Il existe plusieurs manières de générer un labyrinthe. Pour notre exemple, j'ai implémenté l'algorithme dit de la fusion aléatoire de chemins. En effet, il est simple. C'est la méthode **genererLabyrinthe** de la classe **TLabyrinthe** définie dans uLabyrinthe.pas qui réalise les actions suivantes.

Voici la définition de la classe **TLabyrinthe** :

```
TLabyrinthe = class
  tailleX, tailleY, niveauOuverture : integer;
  matrice : array of array of integer;
  listeMurs : TList<TMur>;
private
  procedure ouvrirLabyrinthe;
  procedure listerMursCassables;
```

```
public
  constructor Create(X, Y: integer); virtual;
  destructor Destroy; override;
  function getNbVoisinsValeur(i, j, valeur : integer):integer;
  procedure genererLabyrinthe(complexe : boolean = false);
end;
```

Nous partons d'un tableau de taille n x n initialisé comme indiqué dans la figure 7 (exemple d'un tableau de taille 7x7).

-1	-1	-1	-1	-1	-1	-1
-1	0	-1	1	-1	2	-1
-1	-1	-1	-1	-1	-1	-1
-1	3	-1	4	-1	5	-1
-1	-1	-1	-1	-1	-1	-1
-1	6	-1	7	-1	8	-1
-1	-1	-1	-1	-1	-1	-1

Figure 7

Les cellules à -1 correspondront aux murs (elles ont un fond gris sur la **figure 7**) et les cellules avec un nombre supérieur ou égal à zéro seront des emplacements libres. Voici le code qui génère une telle matrice :

```
setlength(matrice, tailleX, tailleY);
var nb := 0;
for var i := 0 to tailleX-1 do begin
 for var j := 0 to tailleY-1 do begin
  if (i = 0) or (i = tailleX-1) then matrice[i,j] := -1
  else begin
    if (j = 0) or (j = tailleY-1) then matrice[i,j] := -1
    else begin
     if i mod 2 = 0 then matrice[i,j] := -1
     else begin
       if j mod 2 = 0 then matrice[i,j] := -1
       else begin
         matrice[i,j] := nb;
         inc(nb);
       end;
     end;
    end;
  end;
 end;
end;
```

Puis, nous dressons la liste des murs « cassables », c'est-à-dire des murs adjacents à une cellule libre. Attention, les murs du pourtour du labyrinthe ne sont pas à considérer comme cassables. Un mur cassable ne peut être relié qu'à deux cellules libres. La **figure 8** montre les murs cassables en bleu de notre exemple de 7x7.

-1	-1	-1	-1	-1	-1	-1
-1	0	-1	1	-1	2	-1
-1	-1	-1	-1	-1	-1	-1
-1	3	-1	4	-1	5	-1
-1	-1	-1	-1	-1	-1	-1
-1	6	-1	7	-1	8	-1
-1	-1	-1	-1	-1	-1	-1

Figure 8

Voici le code qui permet de lister les murs cassables :

```
procedure TLabyrinthe.listerMursCassables;
begin
 listeMurs.Clear;
 var mur : TMur;
 for var i := 1 to tailleX-2 do begin
  for var j := 1 to tailleY-2 do begin
   if i mod 2 = 0 then begin
    if j mod 2 <> 0 then begin
     mur.x := i;
     mur.y := j;
     listeMurs.Add(mur);
    end;
   end else begin
    if j mod 2 = 0 then begin
     mur.x := i;
     mur.y := j;
     listeMurs.Add(mur);
    end;
   end;
  end;
 end;
end;
```

Une fois cette liste établie, nous itérons tant qu'il reste un élément dans cette liste des murs cassables. À chaque itération, on sélectionne aléatoirement un élément de cette liste. Si les valeurs des deux cellules libres adjacentes sont différentes, alors on casse le mur en lui donnant la valeur d'une des cellules libres adjacentes. Nous affectons également cette même valeur à l'autre case libre adjacente ainsi qu'à toutes les cellules qui ont la même valeur que la seconde cellule libre. On supprime l'élément de la liste.

La **figure 9** illustre cela : la cellule en rouge est le mur cassé, sa valeur passe à 4 (valeur d'une des deux cellules libres adjacentes) et l'autre cellule libre adjacente passe également à la valeur 4.

-1	-1	-1	-1	-1	-1	-1
-1	0	-1	1	-1	2	-1
-1	-1	-1	-1	-1	-1	-1
-1	3	-1	4	4	4	-1
-1	-1	-1	-1	-1	-1	-1
-1	6	-1	7	-1	8	-1
-1	-1	-1	-1	-1	-1	-1

Figure 9

Le code de la boucle est le suivant :

```
while listeMurs.Count > 0 do begin
  var unMur := listeMurs[random(listeMurs.count)];
  var x := unMur.x;
  var y := unMur.y;
  var cell1, cell2 : integer;

  if matrice[x -1, y] < 0 then begin
    cell1 := matrice[x, y -1];
    cell2 := matrice[x, y +1];
  end else begin
    cell1 := matrice[x -1, y];
    cell2 := matrice[x +1, y];
  end;

  if cell1 <> cell2 then begin
    matrice[x,y] := cell1;
    for var i := 1 to tailleX-2 do begin
      for var j := 1 to tailleY-2 do begin
        if matrice[i,j] = cell2 then matrice[i,j] := cell1;
      end;
    end;
  end;

  listeMurs.Delete(listeMurs.IndexOf(unMur));
end;
```

Une fois la liste de murs cassables vide, nous nous retrouvons avec par exemple le tableau de la **figure 10** : notre labyrinthe est créé !

-1	-1	-1	-1	-1	-1	-1
-1	0	0	0	-1	0	-1
-1	0	-1	-1	-1	0	-1
-1	0	0	0	0	0	-1
-1	-1	0	-1	-1	0	-1
-1	0	0	0	-1	0	-1
-1	-1	-1	-1	-1	-1	-1

Figure 10

En générant ainsi notre labyrinthe, nous constatons deux choses :

- pour aller d'une cellule A vers une cellule B, il n'y a toujours qu'un seul chemin possible ;
- le labyrinthe n'est qu'un long couloir : il n'y a pas de salle.

Il s'agit en effet d'un labyrinthe parfait.

Pour remédier à cela, il suffit simplement de casser aléatoirement des murs (mais pas ceux de l'enceinte). Pour notre jeu, j'ai choisi une autre solution. La méthode **genererLabyrinthe** prend un paramètre booléen facultatif. Il est à false par défaut et la méthode génère alors un labyrinthe parfait. Si on passe true en paramètre, alors une étape supplémentaire est effectuée dite d'ouverture du labyrinthe. C'est alors la méthode **ouvrirLabyrinthe** qui est appelée dont voici le code :

```
procedure TLabyrinthe.ouvrirLabyrinthe;
begin
 for var h := 1 to niveauOuverture do
  for var i := 1 to tailleX-2 do
   for var j := 1 to tailleY-2 do
    if matrice[i,j] = -1 then begin
     if getNbVoisinsValeur(i, j, 0) >= 3 then matrice[i,j] := 0;
    end;
end;
```

On parcourt la matrice à la recherche des murs (cellules ayant la valeur -1) entourés d'au moins 3 cellules adjacentes vide (valeur 0). Si tel est le cas on supprime le mur en affectant 0 à cette cellule.

Il est possible des réaliser plusieurs fois ce « nettoyage » en jouant sur la valeur de **niveauOuverture**.

La fonction **getNbVoisinsValeur** permet de renvoyer le nombre de cellules voisines d'une cellule donnée ayant une valeur particulière.

```
function TLabyrinthe.getNbVoisinsValeur(i, j, valeur : integer):integer;
begin
 result := 0;
 if matrice[i+1,j] = valeur then inc(result);
 if matrice[i-1,j] = valeur then inc(result);
 if matrice[i,j+1] = valeur then inc(result);
 if matrice[i,j-1] = valeur then inc(result);
end;
```

La **figure 11** montre le labyrinthe obtenu à partir de la **figure 10** et avec le **niveauOuverture** à 2.

-1	-1	-1	-1	-1	-1	-1
-1	0	0	0	-1	0	-1
-1	0	0	0	-1	0	-1
-1	0	0	0	0	0	-1
-1	0	0	0	0	0	-1
-1	0	0	0	-1	0	-1
-1	-1	-1	-1	-1	-1	-1

Figure 11

Création de la scène 3D

Comme évoqué, l'éditeur d'interface de Delphi permet de constituer facilement l'interface de notre jeu. Je passe rapidement sur les éléments 2D qui sont un **TLayout** en haut de la fenêtre pour afficher les informations (messages, collisions du joueur avec un objet, chronomètre) et un second en bas visible que sur les plateformes mobiles, car il contient les boutons d'interaction.

Concentrons-nous sur la partie 3D et plus précisément les composants placés en tant qu'enfants du composant **TViewport3D**. Nous allons placer via le concepteur graphique les objets 3D présents qu'une seule fois :

- nous plaçons plusieurs **TDummy** qui sont des objets 3D invisibles. À l'image des **TLayout** en 2D, ils permettent de regrouper des objets 3D. Ainsi, en appliquant par exemple une transformation (rotation, translation, mise à l'échelle…) au **TDummy**, tous ses enfants en bénéficient.
  Nous plaçons donc un **TDummy** nommé dmyNiveau qui contiendra tout le niveau (labyrinthe, joueur, interrupteur et la sortie).
  Plaçons un second **TDummy** nommé dmyJoueur qui correspondra à la position du joueur. Celui-ci contient un **TCamera** (le point de vue du joueur) et un autre **TDummy** nommé dmyDirection qui placé de manière à être légèrement décalé du dmyJoueur sur l'axe Z. La différence de position absolue entre le dmyJoueur et le dmyDirection permet de déterminer facilement la direction du joueur. Nous le verrons lorsque nous aborderons les déplacements et les collisions.
  Nous pouvons également utiliser les **TDummy** pour classer les objets 3D. C'est d'ailleurs ce qui est fait dans notre exemple. En effet, le **TDummy** nommé dmyBonus contient les objets bonus (l'interrupteur et la sortie dans notre cas) et le **TDummy** nommé dmyMurs contiendra les murs du labyrinthe. Le fait de classer les objets de la sorte permet de simplifier les interactions et collisions comme nous le verrons par la suite.
  Enfin, nous plaçons un dernier **TDummy** nommé dmyNextPosition. Il permet de calculer la prochaine position du joueur (dmyJoueur). Nous détaillerons cela plus tard.
- le **TPlane** nommé pSol représentera le sol. Nous lui assignons un **TLightMaterialSource** qui correspond au matériau (la texture qui sera appliquée et sa réaction aux sources de lumière) que l'on souhaite donner au sol ;
- le dmyBonus contient un **TCylinder** qui symbolise la porte de sortie du labyrinthe et un **TCube** qui symbolise l'interrupteur ;
- un composant de type **TLight** est placé en tant qu'enfant du **TCamera**. Il s'agit de la source de lumière qui éclaire notre scène 3D. Placée ainsi, elle suivra les déplacements du joueur. Elle est de type spot afin de simuler le fait que le joueur tient dans sa main une lampe torche ;
- enfin, un **TFloatAnimation** est également placé en tant qu'enfant du **TCamera**. Il s'agit d'une petite animation qui déplacera sur l'axe Y (de haut en bas et de bas en haut) la caméra dès que le joueur se déplacera. Ceci pour donner l'impression de marcher.

D'autres composants sont placés via l'éditeur d'interface graphique également. Il s'agit d'éléments en 2D classiques (**TLayout**, **TLabel**, **TRectangle**) qui servent à afficher à l'uti-

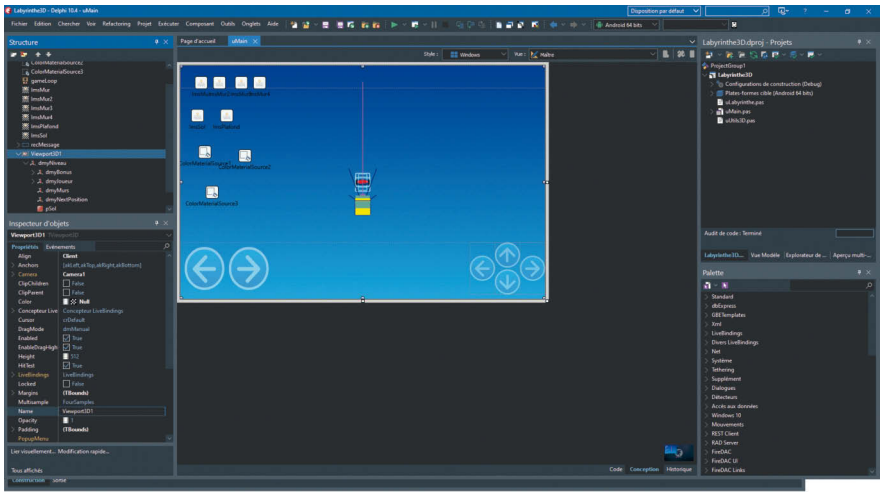


Figure 12

lisateur un bandeau en haut qui indique certaines informations (temps écoulé, collision avec tel objet…) et un bandeau en bas qui n'est visible que sur plateforme mobile qui contient les boutons permettant d'interagir (avancer, reculer, tourner…).

La **figure 12** montre le concepteur d'interface graphique et la vue structure (en haut à gauche de l'image) montre l'arborescence des composants en particulier pour la scène 3D. Concernant le labyrinthe, nous avons étudié précédemment la manière dont il sera généré. Pour transformer cela en labyrinthe 3D, nous allons créer dynamiquement les objets 3D. Cela est fait dans la procédure **creerNouvellePartie** que voici :

```
procedure TMainForm.creerNouvellePartie;
begin
 recMessage.Visible := false;
 niveau := TLabyrinthe.Create(tailleNiveau, tailleNiveau);
 vitesse := 0;
 vitesseRotation := 0;
 arriveeActive := false;
 cArrivee.MaterialSource := ColorMaterialSource3;
 cArrivee.Height := 0.05;
 cInterrupteur.Height := 0.05;
 pSol.Width := tailleNiveau;
 pSol.Height := tailleNiveau;
 pSol.Position.X := -0.5;
 pSol.Position.Z := -0.5;
 niveau.niveauOuverture := 2;
 niveau.genererLabyrinthe(true);
 genererContenuLabyrinthe;
 camera1.Position.Point := Point3D(0,-0.5,-0.6);
 dmyMurs.DeleteChildren;

 for var i := 0 to niveau.tailleX -1 do begin
  for var j := 0 to niveau.tailleY -1 do begin
   case niveau.matrice[i,j] of
    -1: creerMurs(i, j);
    1 : begin
         dmyJoueur.Position.X := i - niveau.tailleX * 0.5;
         dmyJoueur.Position.z := j - niveau.tailleY * 0.5;
        end;
    2 : begin
         cArrivee.position.X := i - niveau.tailleX * 0.5;
         cArrivee.position.Z := j - niveau.tailleY * 0.5;
         cArrivee.position.Y := - cArrivee.Height * 0.5;
        end;
    3 : begin
         cInterrupteur.position.X := i - niveau.tailleX * 0.5;
         cInterrupteur.position.Z := j - niveau.tailleY * 0.5;
         cInterrupteur.position.Y := - cInterrupteur.Height * 0.5;
        end;
   end;
  end;
 end;

 heureDebut := now;
 gameLoop.Start;
end;
```

Cette procédure permet de démarrer une nouvelle partie. Nous procédons à un certain nombre d'initialisations et en particulier nous créons un objet de type **TLabyrinthe** et nous générons un labyrinthe.

Ensuite, nous parcourons les éléments du labyrinthe généré et en fonction de la valeur de chaque élément de la matrice, nous créons l'objet correspondant. Par exemple, si la valeur d'un élément est -1, on crée un objet **TCube** qui fera office de mur. Nous gérons également les positions de départ du joueur, de l'interrupteur et de la sortie.

Enfin, nous récupérons l'heure de début de partie et nous démarrons la boucle principale du jeu.

La création des murs est réalisée via la méthode **creerMurs** dont le code est le suivant :

```
procedure TMainForm.creerMurs(posX, posY : integer);
begin
 var unCube := TCube.Create(nil);
 unCube.Parent := dmyMurs;
 unCube.Name := 'cube' + dmyMurs.ChildrenCount.ToString;
 unCube.Position.Y := -0.5;
 unCube.Width := 1;
 unCube.Height := 1;
 unCube.Depth := 1;
 unCube.Position.X := posX - niveau.tailleX * 0.5;
 unCube.Position.Z := posY - niveau.tailleY * 0.5;
 unCube.TwoSide := true;
 unCube.SubdivisionsDepth := 5;
 unCube.SubdivisionsHeight := 5;
 unCube.SubdivisionsWidth := 5;
 case random(15) mod 7 of
  1: unCube.MaterialSource := lmsMur2;
  2,3 : unCube.MaterialSource := lmsMur;
  4: unCube.MaterialSource := lmsMur4;
  else unCube.MaterialSource := lmsMur3;
 end;
end;
```

Rien de compliqué dans ce code où l'on instancie simplement un objet **TCube**, puis on renseigne quelques-unes de ses propriétés. En particulier, nous lui affectons une texture aléatoirement parmi les quatre disponibles.

BIGDATA & AI PARIS by Xcorp

Conférence et Exposition

10e Edition • 28 & 29 septembre 2021

Palais des Congrès • PARIS

DevOps

Reinforcement learning

Ethique

BIG DATA & AI

TIME TO ACCELERATE

Robotique

Data for Good

Cybersecurité

Gouvernance et qualité

Cloud

IA Green

Machine learning

Automatisation

NLP / NLG

15 000 PARTICIPANTS

200 PARTENAIRES

350 CONFÉRENCES & ATELIERS

Inscrivez-vous sur www.bigdataparis.com

Boucle principale

Tout comme nous l'avons vu dans l'article du n°247 de Programmez!, un jeu vidéo en 3D repose aussi sur une boucle principale dont on ne sort que lorsque le joueur a gagné ou perdu. À chaque itération dans cette boucle, nous devons gérer l'affichage, les déplacements, les collisions, les interactions du joueur…

Nous utiliserons à nouveau dans cet exemple un objet de type **TFloatAnimation**. Il sera nommé **GameLoop** et son événement **OnProcess** servira de boucle principale.

Voici le code de cet événement :

```
procedure TMainForm.gameLoopProcess(Sender: TObject);
begin
 lblChrono.text := formatDateTime('nn:ss', now - heureDebut);

 gererTouches;

 if vitesse = 0 then aniCourse.StopAtCurrent
 else begin
  if not(aniCourse.Running) then aniCourse.Start;
 end;

 var collisionMur := collisionDummyChilds(dmyMurs, dmyNextPosition);
 if not(collisionMur.bool) then begin
  lblInfos.text := '';
  dmyJoueur.Position.Point := dmyNextPosition.Position.Point;
  var collisionBonus := collisionDummyChilds(dmyBonus, dmyJoueur);
  if collisionBonus.bool then gererCollisionBonus(collisionBonus.objet);
 end else begin
  lblInfos.text := 'Collision avec ' + collisionMur.objet.Name;
  vitesse := 0;
 end;
end;
```

Dans cet événement, nous commençons par mettre à jour l'affichage du chronomètre. Il s'agit dans notre exemple d'une simple indication du temps mis par le joueur pour sortir du labyrinthe. Il est possible d'améliorer le gameplay en gérant un tableau des meilleurs temps ou de gérer un temps limite par exemple.

Ensuite, nous invoquons la méthode **gererTouches** qui permet de répondre aux commandes que le joueur passe. Nous détaillerons cela un peu plus loin dans l'article.

Nous passons alors à la gestion d'une petite animation nommée **aniCourse** et de type **TFloatAnimation**. Cette animation est enfant du **TCamera** et agira sur sa position sur l'axe Y (sa hauteur). Si la vitesse du joueur est à zéro, alors on stoppe l'animation, sinon on la démarre. Cela aura pour effet de faire monter et descendre légèrement la caméra lorsque le joueur se déplace afin de donner l'impression qu'il court.

Enfin, nous gérons les collisions du joueur avec les murs et avec les bonus. Nous verrons cela plus tard. Si le joueur entre en collision avec un mur, alors nous passons sa vitesse à zéro et nous affichons le nom de l'objet avec lequel il est entré en collision.

Gestion des déplacements

Lors de la création de la scène 3D, nous avons placé un **TDummy** nommé dmyJoueur correspondant à la position du joueur. Nous avons également placé un autre **TDummy** nommé dmyNextPosition. Ce dernier va servir dans la gestion des déplacements du joueur. En effet, nous allons l'utiliser pour déterminer la prochaine position du joueur à partir de sa position actuelle (dmyJoueur), de son orientation et de sa vitesse. Ensuite, nous allons tester cette future position afin de contrôler qu'elle soit valide. Pour être valide, la position future ne doit pas entrer en collision avec un mur du labyrinthe. De même, nous testons si la position future est en collision avec un bonus (l'interrupteur ou la sortie). Si tel est le cas alors nous réalisation l'action correspondante.

Après ces contrôles, si la position future est possible, alors nous affectons cette position future à la position du dmyJoueur. Si non, on laisse le dmyJoueur à sa position.

Gestion des collisions

Dans notre jeu, les différents éléments (joueur et obstacles) sont représentés par des cubes (**TDummy** ou **TCube**) donc des objets 3D simples. La méthode de détection des collisions dite « bounding box » que nous allons utiliser sera suffisante. Pour détecter une collision entre deux objets 3D, nous allons en fait déterminer s'il y a collision entre les deux boîtes englobantes des objets. La **figure 13** montre un arbre et une voiture modélisés en 3D. Les boîtes englobantes sont symbolisées par les rectangles 3D en vert.

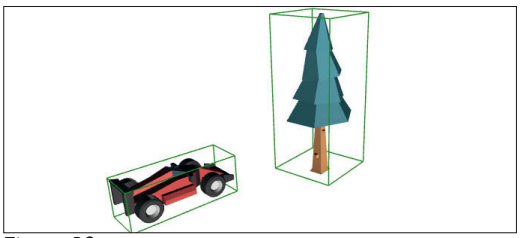


Figure 13

Nous détecterons une collision si une partie du rectangle autour de la voiture entre dans le rectangle autour de l'arbre. Nous constatons aisément sur cet exemple que la méthode n'est pas précise. En effet, l'objet 3D n'occupe pas tout le volume de la boîte. Du coup, la méthode va détecter une collision entre les boîtes alors que les objets réels ne se touchent pas. Dans notre cas, cela n'est pas gênant, car les objets manipulés sont des cubes (et donc des rectangles). En effet, les murs sont représentés par des **TCube** et le joueur par un TDummy. La boîte englobante de chaque objet aura le même volume que l'objet. Dans ces conditions, la méthode des bounding box est suffisante et intéressante, car simple à implémenter.

Nous avons vu que sous FMX, un objet 3D avait une propriété nommée position qui correspond à la position du centre de l'objet. De plus, ses propriétés height (hauteur), width (largeur) et depth (profondeur) « s'étalent » de part et d'autre du centre de l'objet.

Passons un instant en deux dimensions pour matérialiser une collision sur l'axe des abscisses. La **figure 14** montre deux rectangles avec leurs centres C1 et C2.

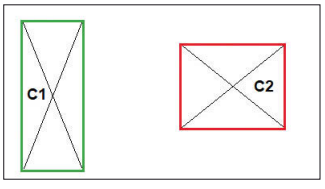


Figure 14

Imaginons que ces rectangles se déplacent sur l'axe des abscisses en sens inverse l'un de l'autre. La figure 15 représente ces mêmes rectangles en collision.

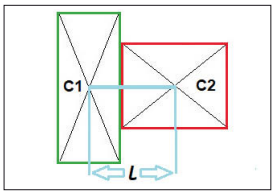


Figure 15

Nous constatons alors que la distance entre les centres C1 et C2 (distance ***L***) est égale à la somme des moitiés des largeurs des deux rectangles.

Nous allons donc faire de même sur les axes des ordonnées (Y) et l'axe des cotes (Z).

L'implémentation de cela est faite dans la méthode **collisionEntre2Objets** définie dans l'unité uUtils3D.pas. Voici son code :

```
function collisionEntre2Objets(objet1, objet2 : TControl3D): TGBECollisionRetour;
begin
 result.objet := nil;
 result.bool := false;
 var DistanceEntreObjets := objet1.Position.Point - objet2.Position.Point;
 var distanceMinimum := (SizeOf3D(objet1) + SizeOf3D(objet2)) * 0.5;

 if ((Abs(DistanceEntreObjets.X) < distanceMinimum.X) and (Abs(DistanceEntre
Objets.Y) < distanceMinimum.Y) and
   (Abs(DistanceEntreObjets.Z) < distanceMinimum.Z)) then begin
  result.bool := true;
  result.objet := objet2;
 end;
end;
```

Cette fonction prend en paramètre deux objets de type **TControl3D** (le type d'objet 3D le plus ancêtre) renvoie un objet de type **TGBECollisionRetour** qui est un simple record contenant un booléen (à true s'il y a collision) et un **TControl3D** qui permet de récupérer l'objet heurté.

Dans cette fonction, nous commençons par calculer la distance entre les centres des deux objets. Ensuite, nous calculons la distance minimum entre les deux objets : une distance inférieure à cette distance minimum signifie qu'il y a collision. Pour ce faire, nous additionnons les tailles des deux objets (sur les 3 axes). Comme vu précédemment, nous multiplions par 0,5 pour obtenir la moitié de la taille des deux objets cumulés.

Enfin, nous comparons pour chaque composante X, Y et Z si la distance calculée entre les deux objets est inférieure à la distance minimum. La fonction sizeOf3D, utilisée dans le calcul de la distance minimum entre deux objets, permet de récupérer la taille de l'objet passé en paramètre sous forme de **TPoint3D**. Voici son code :

```
function SizeOf3D(const unObjet3D: TControl3D): TPoint3D;
begin
 Result :=NullPoint3D;
 if unObjet3D <> nil then
  result := Point3D(unObjet3D.Width, unObjet3D.Height, unObjet3D.Depth );
end;
```

Nous avons maintenant une fonction qui détecte les collisions entre deux objets. Il nous reste à appeler cette méthode pour contrôler les collisions entre le **dmyJoueur** et les différents obstacles.

Lors de la constitution de la scène 3D, nous avons prévu le **dmyMurs** qui contient tous les murs du labyrinthe. Il nous suffit donc de boucler sur les enfants du **dmyMurs**. Pour ce faire, toujours dans l'unité uUtils3D.pas, nous trouvons la méthode collisionDummyChilds.

```
function collisionDummyChilds(aDummy: TDummy; objet3D : TControl3D): TGBE
CollisionRetour;
begin
 var resultat : TGBECollisionRetour;
 resultat.bool := false;
 resultat.objet := nil;
 for var obj in aDummy.Children do begin
  if (obj as TControl3D).visible then begin
   resultat := collisionEntre2Objets(objet3D, (obj as TControl3D));
   if resultat.bool then break;
  end;
 end;
 result := resultat;
end;
```

Cette méthode prend en paramètre un **TDummy** et un **TControl3D**. C'est elle qui va donc contrôler s'il y a collision entre l'objet passé en paramètre et les enfants du **TDummy** passé en paramètre. Pour cela, nous bouclons simplement sur les enfants du **TDummy** et à chaque itération, nous appelons la fonction collisionEntre2Objets. Si une collision est détectée, nous sortons de la boucle.

La fonction collisionDummyChilds est appelée à deux reprises dans la boucle principale : une première fois avec le **dmyMurs** pour détecter les collisions avec les murs et une seconde fois avec **dmyBonus** pour détecter les collisions avec les bonus. Ainsi, si le joueur entre en collision avec l'interrupteur, nous pouvons activer la sortie. Si le joueur entre en collision avec la sortie sans l'avoir activée, rien ne se passe. Si la sortie est activée alors c'est la fin de la partie, le joueur a gagné.

Gestion des touches

Pour se déplacer, le joueur peut utiliser les touches suivantes du clavier :

- Flèche droite ou touche D pour aller vers la droite ;
- Flèche gauche ou touche Q pour aller vers la gauche ;
- Flèche haut ou touche Z pour avancer ;
- Flèche bas ou touche S pour reculer ;
- Touche A pour effectuer un déplacement latéral sur la gauche ;
- Touche E pour effectuer un déplacement latéral sur la droite ;
- Touche F1 pour obtenir une petite aide : l'interrupteur et la sortie auront alors une hauteur importante permettant de les apercevoir au-dessus des murs.

Pour gérer cela, nous utilisons les événements OnKeyDown (appuie sur une touche) et OnKeyUp (l'utilisateur relâche la touche) de la fiche FormMain.

Petite astuce : dans ces événements nous nous contentons de

positionner des flags de type boolean en fonction des touches appuyées. Les actions correspondantes seront réalisées dans la boucle principale via l'appel à la méthode **gererTouches** dont voici le code :

```
procedure TMainForm.gererTouches;
begin
 if toucheDroite or toucheGauche then begin
  if vitesseRotation < vitesseRotationMax then vitesseRotation := vitesseRotation + 0.05;
  if toucheDroite then dmyJoueur.RotationAngle.Y := dmyJoueur.RotationAngle.Y + vitesse
Rotation
  else dmyJoueur.RotationAngle.Y := dmyJoueur.RotationAngle.Y - vitesseRotation;
 end;
 if toucheHaut or toucheBas or toucheDecalageGauche or toucheDecalageDroite
then begin
  if vitesse < vitesseMax then vitesse := vitesse + 0.005;
  if toucheHaut then dmyNextPosition.position.Point := dmyJoueur.position.Point - get
Direction * vitesse;
  if toucheBas then dmyNextPosition.position.Point := dmyJoueur.position.Point + get
Direction * vitesse;
  if toucheDecalageGauche then dmyNextPosition.position.Point := dmyJoueur.position
.Point + getDirection.Rotate(Point3D(0,1,0), Pi * 0.5) * vitesse;
  if toucheDecalageDroite then dmyNextPosition.position.Point := dmyJoueur.position.Point + get
Direction.Rotate(Point3D(0,1,0), -Pi * 0.5) * vitesse;
 end;
end;
```

Cette astuce permet de ne pas remplir le buffer clavier en cas de traitement long à réaliser dans les actions.
De plus, le fait de passer par des booléens lors d'appuis sur les touches permet très facilement d'autoriser d'autres moyens d'interagir. Par exemple, sur Android ou IOS, il n'y a pas de clavier. Nous avons prévu d'afficher pour les plate-formes mobiles le **TLayout** layHMMobile qui contient des boutons. Il suffit alors de positionner ces mêmes booléens lorsque l'utilisateur clique ou appuie sur les boutons.
Détaillons rapidement cette procédure gererTouches. Tout d'abord, si les flèches droites et gauches sont enfoncées, on effectue une rotation du **dmyJoueur** autour de l'axe Y. L'angle de cette rotation est donné par la variable **vityesseRotation** qui permet une petite inertie.
Ensuite, si les autres touches sont enfoncées, cela signifie que le joueur veut se déplacer. Nous allons donc jouer sur la position du **dmyNextPosition**. En effet, gererTouche est invoquée dans la boucle principale avant de tester les collisions : nous calculons donc la future position du joueur (**dmyNextPosition**) et nous vérifions si cette nouvelle position est possible ou non. Si la future position est possible, alors on affecte la position du **dmyNextPosition** à la position du joueur (**dmyJoueur**) sinon, le joueur conserve sa position et ne peut pas se déplacer.
La future position du joueur dépend de sa vitesse de déplacement et de son orientation. La fonction getDirection permet de déterminer l'orientation du joueur. Voici son code :

```
function TMainForm.GetDirection: TPoint3D;
begin
 result := (dmyDirection.AbsolutePosition - dmyJoueur.AbsolutePosition).Normalize;
end;
```

L'orientation du joueur est simplement fournie par la différence de position entre le **dmyDirection** et le **dmyJoueur**. Lorsque l'utilisateur avance ou recule, le déplacement doit être réalisé dans cette direction.
Pour les déplacements latéraux en revanche, il faut que le déplacement ait lieu de manière perpendiculaire à cette direction. C'est pourquoi on invoque la méthode Rotate du TPoint3D en lui indiquant que l'on souhaite lui appliquer une rotation autour de l'axe Y (Point(0,1,0) et d'angle $\pi/2$ en radian.

Le jeu final

Nous voici au terme des explications techniques sur ce petit jeu de labyrinthe en 3D, il est temps de vous montrer le résultat final. (Voir **figure 16**)

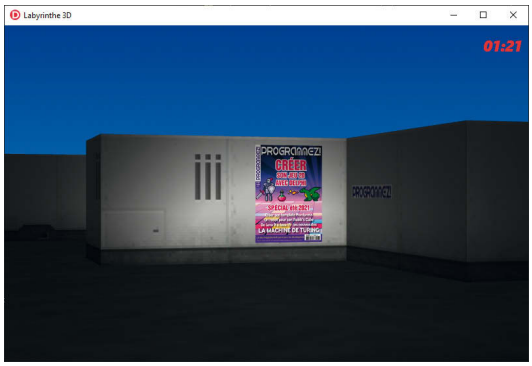


Figure 16

Vous trouverez le code source complet du projet ainsi que les binaires pour Windows, macOS, Linux et Android sur mon Github : https://github.com/gbegreg/Labyrinthe3D.

D'autres exemples

Je vous invite à consulter mon GitHub (https://github.com/gbegreg) qui contient plusieurs projets exemples réalisés en 3D avec FMX, ou des jeux réalisés lors de gamejams (par exemple ma participation à la Global Game Jam 2021 : https://github.com/gbegreg/LostNFound c'est un petit FPS développé de A à Z seul en 48h). Vous y trouverez également la librairie GBE3D gratuite, open source et fournie avec de nombreuses démonstrations (https://github.com/gbegreg/GBE3D).

Conclusion

Nous n'avons fait qu'effleurer les possibilités qu'offre la 3D dans cet article. J'espère que cette petite introduction à la 3D multiplateforme avec Delphi vous aura plu et aura titillé votre curiosité. Bon amusement !

CONSEIL - FORMATION
EXPERTISE À LA DEMANDE - PRESTATION

DEV-SEC-OPS

CONTACT@FGTECH.FR WWW.FGTECH.FR

MARDI 14 SEPTEMBRE
À MARSEILLE

WAX

CONFÉRENCE 100% REX EN PLEIN AIR

PRÉSENTIEL & STREAMING

www.waxconf.fr

UN ÉVÉNEMENT IMAGINÉ ET ORGANISÉ PAR Les Filles & Les Garçons DE LA TECH

Comparatif des outils « no code » de Microsoft pour la Computer Vision

La vision par ordinateur (Computer Vision ou CV) est une discipline qui fascine en tant que substitut à l'œil humain pour comprendre et décoder des images. Analyse d'une route en temps réel, diagnostic médical, reconnaissance faciale en sont quelques exemples bien connus. Aujourd'hui, des outils simples et épatants d'efficacité sont à la portée de tous.

Paul PETON

manager expert Data Science,
paul.peton&@avanade.com,
https://methodidacte.org

Passionné par les chiffres sous toutes leurs formes, Paul assiste ainsi les entreprises dans la mise en place d'une démarche MLOps, allant de l'émergence des projets de valorisation des données jusqu'à leur industrialisation, en particulier dans le cloud. Nommé MVP en Intelligence Artificielle par Microsoft depuis 2018, puis sur la Data Platform en 2020, il anime et participe à plusieurs communautés autour de la data, dont le meetup Azure Nantes.

Dans les années 2010, l'apprentissage automatique par réseaux de neurones profonds (*Deep Learning*) a bouleversé le domaine de la vision par ordinateur en donnant des résultats exceptionnels. Par la suite, les *frameworks* de Deep Learning (TensorFlow, Keras, Caffee, PyTorch, etc.) ont permis aux développeurs de prendre en main de manière quasi industrielle cette nouvelle approche. Il manquait alors des outils plus accessibles aux populations appétentes à l'approche dite « *no code / low code* », c'est-à-dire ne nécessitant pas ou peu de code.

C'est en particulier la définition de l'architecture du réseau de neurones profond qui est ardue. Celle-ci se base pour les réseaux dits convolutifs sur l'empilement de plusieurs couches réalisant successivement l'extraction des caractéristiques de l'image puis la reconnaissance de la classe associée. Un tel réseau doit s'entraîner longtemps et sur un grand volume de données pour devenir performant. Mais ces modèles, une fois entraînés, pourront être réutilisés, sans devoir tout reprendre de zéro !

L'offre Microsoft pour la vision par ordinateur

Avec l'émergence du cloud public, Microsoft a mis à disposition des services cognitifs répondant à de nombreux besoins de traitement et d'analyses des **données non structurées** (images, sons, vidéo, texte brut…). Dans le domaine de la vision, les services proposés par l'éditeur couvrent (https://azure.microsoft.com/fr-fr/services/cognitive-services/computer-vision/) :

- L'extraction de textes depuis des images (OCR)
- La compréhension d'images (détection d'objets, reconnaissance de visages, d'émotions…)
- L'analyse spatiale (présence et mouvement de personnes dans un espace)

Ces services utilisent des modèles **déjà entraînés,** mais sur des corpus de données très génériques. Dans la plupart des cas, il n'était pas possible de fournir de nouvelles données d'entraînement pour passer d'un cadre générique à un cadre plus spécifique. Or, c'est souvent sur des cas d'usage très précis qu'une organisation peut dégager de la valeur.

C'est donc dans ce cas de figure que nous avons besoin d'un **outil d'entraînement des modèles** réalisant les opérations suivantes :

- Acquisition et labellisation des images
- Entraînement du modèle, idéalement en profitant de couches génériques déjà entraînées
- Test et évaluation du modèle
- Exposition du modèle pour réaliser des prévisions
- Capacité à exporter le modèle pour l'utiliser dans des environnements de production

Deux solutions sont aujourd'hui disponibles et l'objet de cet article est de les évaluer.

Le premier outil, apparu en 2018, est un service cognitif Microsoft disposant d'un portail dédié : **Custom Vision**, accessible sur https://www.customvision.ai/. **Figure 1**

Le second, **Lobe**, est issu de l'acquisition d'une entreprise spécialisée dans ce domaine, datant de 2018. Pour autant, Lobe n'est devenu disponible qu'au second semestre 2020, en version Beta. C'est un exécutable que l'on télécharge (https://lobe.ai/) et qui s'installe sur le poste local. Une inscription par adresse mail sera nécessaire sur laquelle vous recevez les annonces des mises à jour du produit. Nous sommes donc ici face à un outil de type client lourd, qui utilisera les capacités de la machine sur laquelle il tourne. **Figure 2**

Avant de comparer les deux outils, nous devons noter que les tâches couvertes ne sont pas tout à fait les mêmes. À ce jour (juillet 2021), Lobe ne fait que de la classification d'images et non de la détection d'objets. Custom Vision dispose de ces deux possibilités. Lobe ne permet de plus qu'une seule étiquette (*label*) par image.

Dans une pratique « *no code* » et pour mieux comparer les

Figure 1

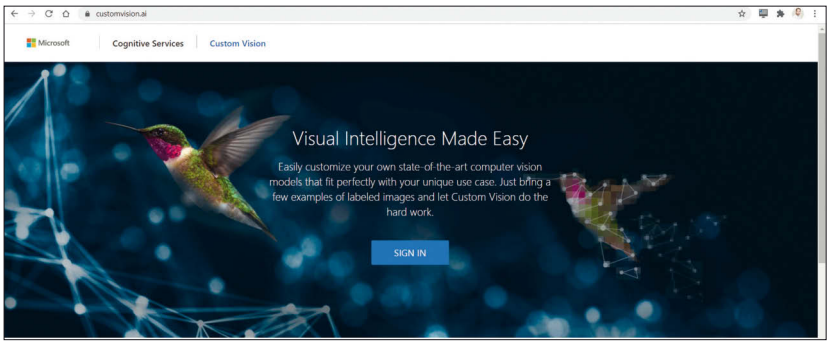
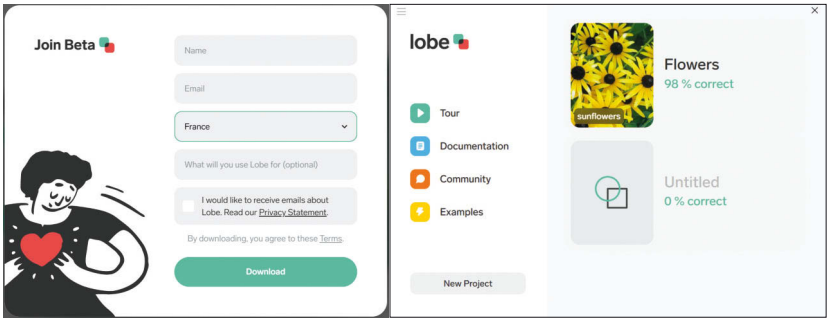


Figure 2



approches, nous n'évoquerons pas ici l'API REST de Custom Vision, ni l'un des différents SDKs qui permettraient d'optimiser en particulier les tâches de chargement et labellisation des données.

Nous ne comparerons donc qu'un scénario de classification d'images, multi-classe et non multi-label, en partant du jeu de données « **flowers** » disponible auprès de l'Université de Harvard. Il s'agit de plus de 3600 fichiers JPG des variétés de fleurs suivantes : *daisy* (633 fichiers), *dandelion* (898), *roses* (641), *sunflowers* (699), *tulips* (799).

Le jeu d'entraînement est disponible sur https://dataverse.harvard.edu/file.xhtml?fileId=4105627&version=8.0 et le jeu de test sur https://dataverse.harvard.edu/file.xhtml?fileId=4159750&version=8.0

Comparaison détaillée des fonctionnalités

Chargement et labellisation des images

Avec Lobe, un minimum de 5 images par label est nécessaire pour débuter l'entraînement. Dans la pratique, ce nombre doit être bien entendu plus élevé pour améliorer les performances. **Figure 3**

Trois modes d'import sont possibles :

- L'acquisition par caméra et capture d'écran
- L'import d'images locales (à labelliser plus tard)
- L'import de répertoires locaux dont le nom définira le label de l'image

Ce dernier mécanisme est bien plus efficace ! Il progresse par lots d'environ 50 images par sous-répertoire. Nous chargeons et labellisons donc les données en moins de deux minutes. Outre le fait d'équilibrer les classes, il est recommandé de varier les arrière-plans, la luminosité ou encore le niveau de zoom. L'entraînement démarre automatiquement et nous pouvons suivre sa progression en temps réel. Le jeu de données est partagé entre 80% pour l'entraînement et 20% pour la validation du modèle. Cette répartition n'est aujourd'hui pas paramétrable.

Pour Custom Vision, il faut, avant de se lancer, provisionner une ressource Cognitive Services depuis le portail Azure. Nous pouvons ensuite définir le projet souhaité au moyen de la boîte de dialogue ci-contre. **Figure 4**

Nous choisissons ici le type de projet, puis le sous-type s'il s'agit d'une classification, puis un domaine éventuel se rapprochant du cas d'usage mis en œuvre (prendre General si aucun domaine ne semble convenir). Il s'agit ici d'une approche de **Transfer Learning**, permettant de bénéficier de couches déjà entraînées dans le réseau de neurones profond. Les domaines dits « **compacts** » sont des versions plus légères de ces couches, sans doute moins performantes, mais dont l'utilisation permettra plus tard l'export du modèle. Le chargement des images se fait depuis le poste local et il est possible de *tagger* dans la foulée l'ensemble d'images importées. Pour les cinq labels souhaités, nous devons donc reproduire autant de fois l'opération de chargement. **Figure 5**

L'interface peut signaler si une image a déjà été importée. Nous ne disposons toutefois pas du nom des fichiers en doublons. La recommandation de l'éditeur est de charger au moins 50 images par label et de ne pas dépasser un ratio de 1 pour 2 dans l'équilibre des labels. Il est possible d'ajouter des « images

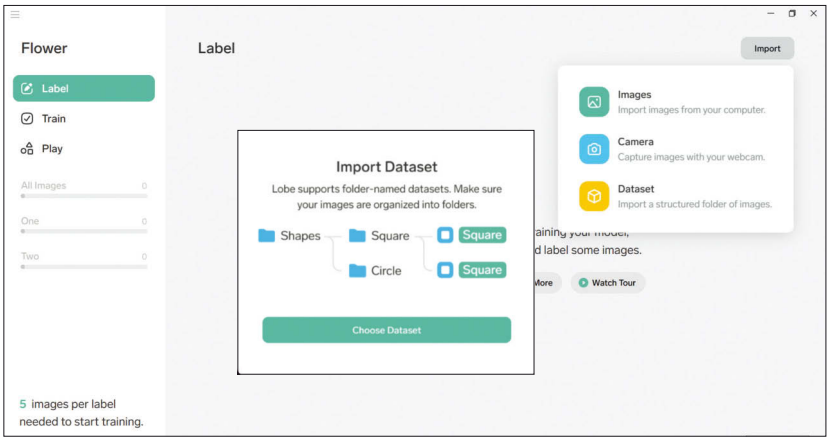


Figure 3

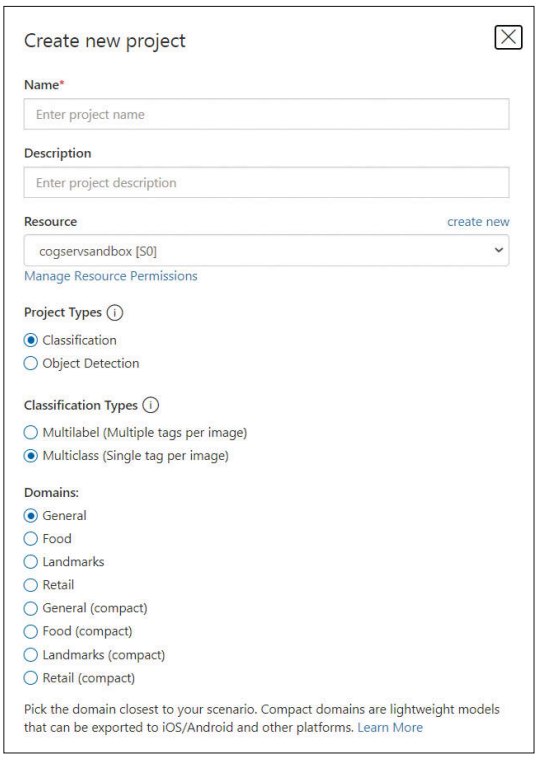


Figure 4

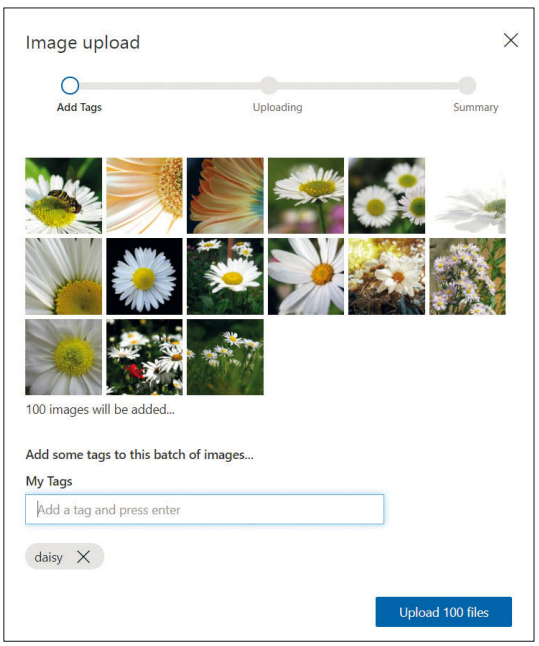


Figure 5

négatives » portant le label spécifique *Negative*, dans le but d'aider le modèle à comprendre ce qu'il ne faut pas rechercher. De telles images devront obtenir un score de 0% pour chaque autre label. Cette astuce fait partie des techniques d'optimisation données dans la documentation officielle.

Entraînement du modèle

Pour Lobe, à partir des paramètres du projet, nous pouvons choisir si le modèle doit être optimal en termes de précision ou bien en temps de calcul. Par défaut, c'est la précision qui est sélectionnée.

En arrière-plan, c'est le réseau neuronal de départ qui va changer en fonction du choix : **ResNet-50** pour la précision et **MobileNetV2** pour la rapidité. L'entraînement débutera dès la fin de l'import des données. Avec le jeu de données *Flowers*, nous obtenons ici un score d'images bien classées de 97%. L'entraînement se relancera à l'ajout de nouvelles images labellisées ou suite à la correction de labels. **Figure 6**

Dans un second temps, il est possible d'optimiser le modèle en cliquant tout simplement sur le bouton « Optimize » (nous ne savons pas ici quels sont les traitements réalisés en arrière-plan). Il n'y alors pas de temps limite et l'optimisation durera tant que des gains de performance sont obtenus. En observant les performances de la machine local, le processeur GPU dont nous disposons ne semble pas sollicité.

Pour Custom Vision, nous disposons de deux modes d'entraînement : *quick training* ou *advanced training*. Un entraînement complet est appelé une **itération**. Chaque itération disposera ensuite de propriétés relatives à l'inférence, c'est-à-dire à la prévision sur de nouvelles images.

Une fois l'entraînement terminé, nous disposons de trois métriques d'évaluation, toujours obtenues ici avec le jeu de données *Flowers* :

- Précision : proportion des prévisions correctes de classes parmi les prévisions affectées à une classe
- Recall (ou rappel) : proportion d'images d'une classe correctement identifiées
- AP pour Average Precision, résumant les deux métriques précédentes **Figure 7**

Un résumé des métriques détaillées par label sera également disponible. Le seul élément paramétrable ici est le seuil (*threshold*) de probabilité permettant à un label d'être attribué (à 50% par défaut).

Sur une deuxième itération réalisée avec un maximum de 7h, nous obtiendrons un très léger gain sur les différentes métriques. Toujours sur le domaine General, mais non compact, nous gagnons environ 1 point de plus sur chaque métrique, atteignant ainsi les scores obtenus avec Lobe.

Test et évaluation du modèle

Depuis l'interface de Lobe, nous basculons sur le menu « Play ». À nouveau, la caméra est disponible pour détecter les labels en temps réel. Il est bien sûr possible d'importer des images sur lesquelles nous pourrons valider ou invalider la prévision réalisée. **Figure 8**

Sous Custom Vision, nous disposons d'une première interface simple dite « quick test » pour télécharger une image locale ou bien soumettre une URL Web. Les fichiers doivent être au format .jpg, .png ou .bmp et ne pas dépasser une taille de 4Mo. Les différents tags sont alors probabilisés. **Figure 9**

Exposition du modèle

À des fins de tests uniquement, nous pourrons réaliser un appel local à l'API REST de Lobe. Un exemple de code va être proposé dans l'un des nombreux langages disponibles : JSON, C, C#, Go, Java, JavaScript, PHP, Python ou encore Shell. Pour utiliser le point de terminaison prédictif, avec un formalisme JSON, nous devons convertir au préalable l'ima-

Figure 6

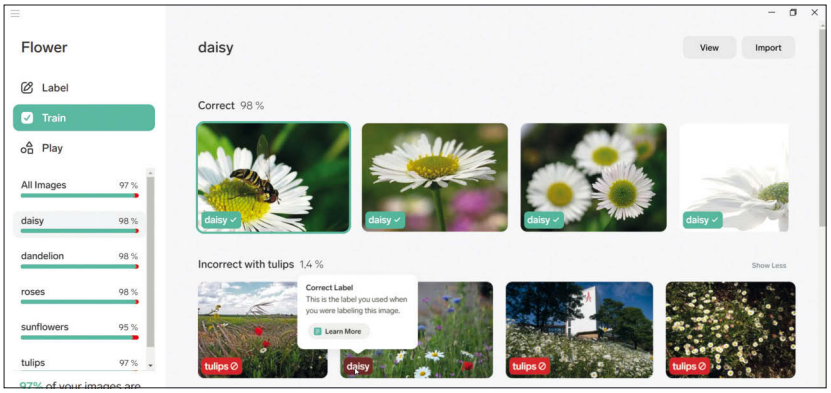


Figure 7

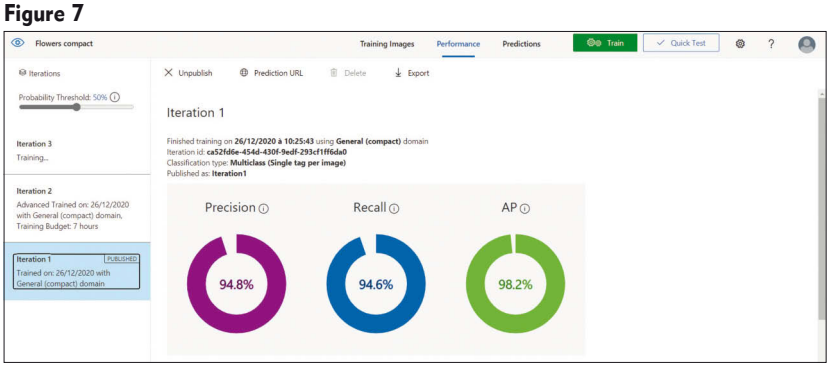
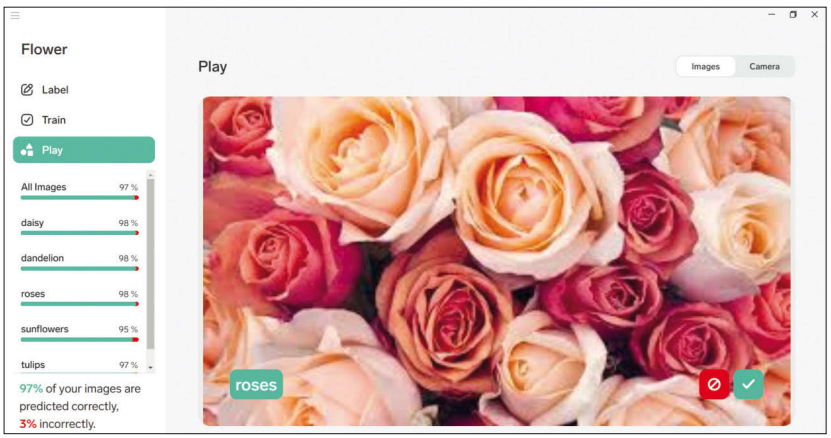


Figure 8



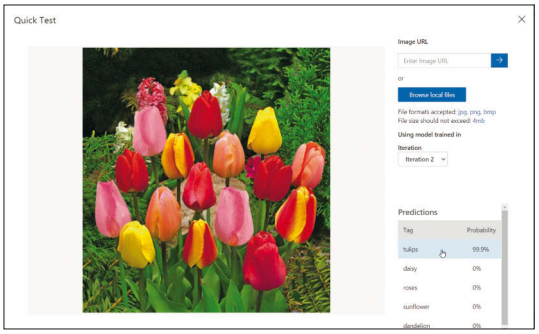


Figure 9

ge à tester en base 64, ce qui peut être fait à l'aide de ce site : https://www.base64-image.de/ Il faudra retirer le début de la chaîne obtenue : ''. Nous pouvons alors réaliser un premier test avec le client Postman, en utilisant la partie Body, au format JSON. **Figure 10**

Le résultat est donné au format JSON. Nous obtenons la probabilité associée à chaque label. Un traitement sera ensuite nécessaire pour identifier automatiquement le label le plus probable.


```
{
  "outputs": {
    "Labels": [
      ["daisy", 0.000005757092367275618],
      ["dandelion", 0.0006570933037437499],
      ["roses", 0.000125496371765621],
      ["sunflowers", 0.10419294983148575],
      ["tulips", 0.8950187563896179]
    ],
    "Prediction": ["tulips"]
  }
}
```


Pour sortir de l'utilisation locale, nous devrons passer par l'export du modèle (voir ci-après).

Sous Custom Vision, afin de disposer d'un point de terminaison pour l'inférence du modèle, nous devons tout d'abord **publier** celui-ci. Nous obtenons alors les informations nécessaires pour utiliser le point de terminaison prédictif. Celui-ci s'appuie sur l'API REST version 3.0 de Custom Vision. **Figure 11**

Les deux modes (URL et fichier image) sont ici aussi disponibles. La clé d'authentification est rappelée, c'est la première valeur (KEY 1) figurant au niveau de la ressource Azure.

Nous pouvons également utiliser le client Postman pour un premier test de ce point de terminaison, à l'aide d'une requête POST, en renseignant les deux en-têtes Prediction-Key et Content-Type. Dans l'approche utilisant un fichier image, celui-ci peut être téléchargé à partir de l'onglet Body, avec l'option *binary*. **Figure 12**

De nouveau, nous obtenons la probabilité associée à chaque label. Il sera donc ici aussi nécessaire d'écrire un traitement de *parsing* de la sortie pour identifier automatiquement le label le plus probable.

En résumé

La base d'un jeu de données commun, il n'a pas été possible de différencier les outils en termes de qualité de modèle. Lobe semble toutefois plus rapide pour entraîner le meilleur modèle, mais il est difficile de comparer un produit tournant localement et en tâche de fond à un service Web dont les ressources de calcul ne sont pas communiquées. **Voir tableau ci-contre**

Focus sur le coût d'utilisation

L'utilisation du client local Lobe, à ce jour en version bêta, est gratuite ! Les ressources d'entraînement utilisées sont votre machine et il n'y a pas besoin d'autre investissement. Le cas échéant, il sera intéressant de se munir d'une webcam externe. Une version commerciale sera vraisemblablement mise sur le marché. D'ici là, en version *public preview*, un support ne peut être exigé de la part de l'éditeur.

Pour Custom Vision, le coût est associé à l'option de tarifica-

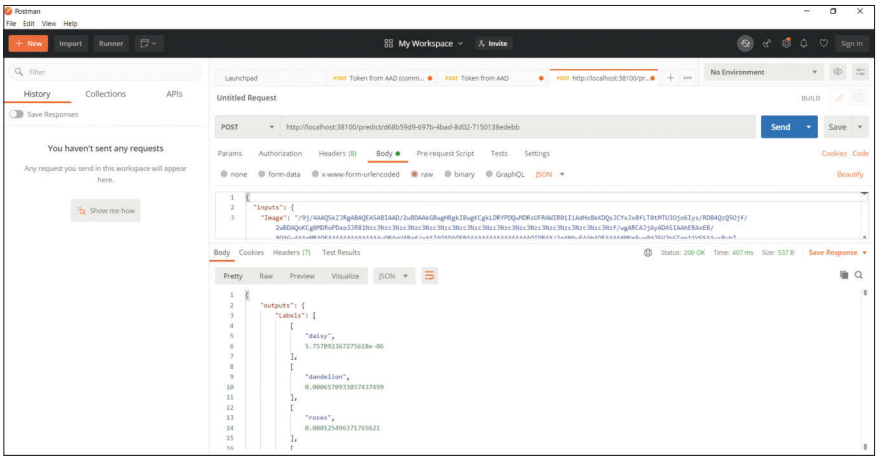


Figure 10

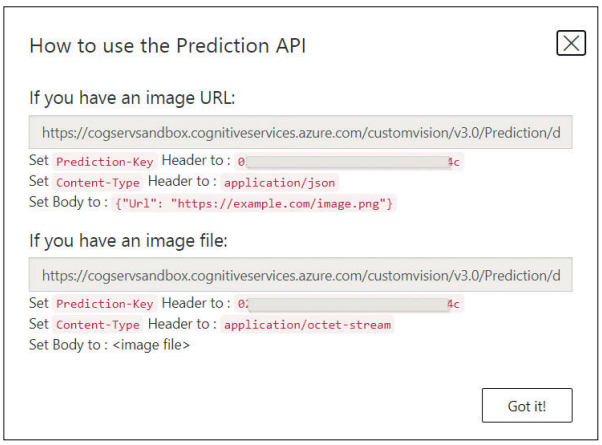


Figure 11

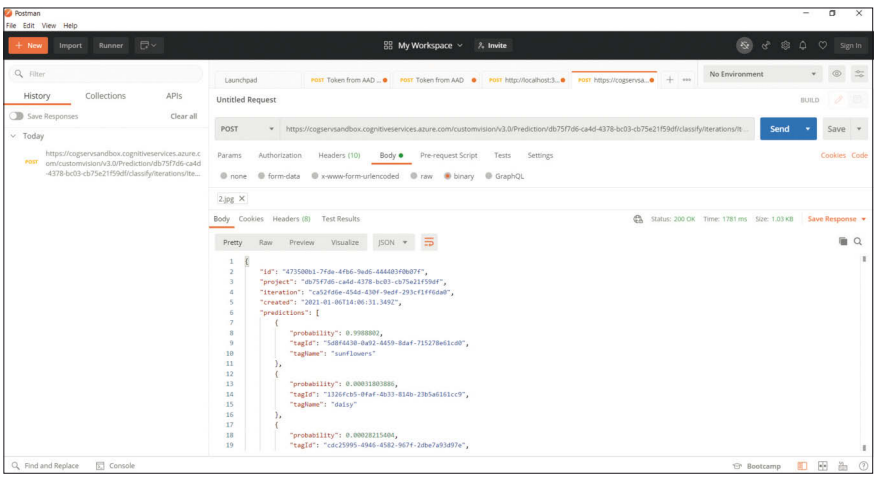


Figure 12

	Microsoft Lobe	Custom Vision
Chargement des images	Upload de dossiers locaux	Upload de dossiers locauxAutomatisation par SDK
Labellisation des images	Sur nommage des dossiers	ManuelDynamique ensuite par « Smart Labeller »
Entraînement du modèle	En tâche de fondUtilisation GPU ?	RapideLong (limité en heures et facturé)
Transfer Learning	ResNet-50 ou MobileNetV2	Depuis un domaine présélectionné
Test et évaluation du modèle	Import localCaméra	Import localURL Web
Exposition du modèle	API locale	Point de terminaison (facturé en production)
Export du modèle	Modèle TensorFlow (mobile, Web ou Python)Modèle ONNX+ script Python d'inférence	Seulement sur domaine compact Modèle ONNX Image Docker pour mise en production avec Azure ML
Performances sur le dataset Flowers	Accuracy : 97%	Accuracy : 97%Modification du seuil de décision
Coût d'utilisation	Gratuit en bêta	Gratuit (F0) ou payant (S0) principalement pour l'entraînement

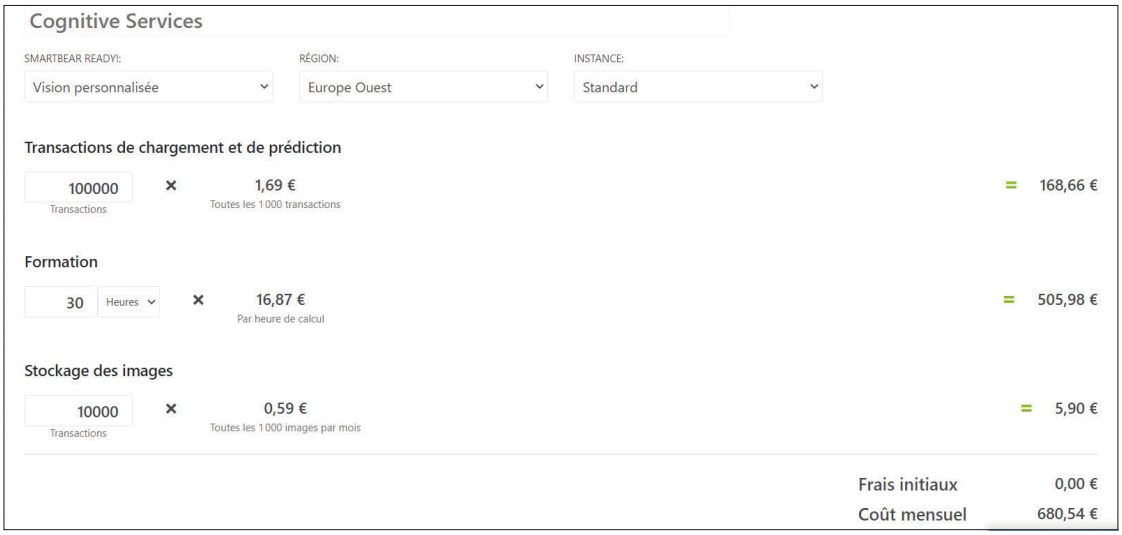


Figure 13

tion définie lors de la création du projet. La calculatrice Azure permet de comparer plus en détail les deux modes possibles : gratuit (F0) et standard (S0). En F0, seuls deux projets peuvent être conservés contre 100 en S0. De plus, l'instance ne peut être utilisée qu'à des fins de test et non pour une mise en production du modèle. Une limite maximum, en heures, peut être donnée lors de l'entraînement, permettant de mieux contrôler les coûts de cette étape. **Figure 13**

Comparaison des modèles obtenus

Nous sommes ici devant 2 outils différents puisque basés sur TensorFlow pour Lobe contre Microsoft Research pour Custom Vision. À l'usage, les outils se révèlent très simples et très efficaces ! Sur le seul critère comparable de la précision (*accuracy*), Lobe dépasse de presque 2 points Custom Vision, en utilisation gratuite, mais il est important d'avoir plusieurs métriques à disposition, ce que ne propose que Custom Vision.

Côté export du modèle, les deux outils se retrouvent sur le **format ONNX**, qui signifie Open Neural Network eXchange et correspond à la volonté de définir un format standardisé et ouvert entre les *frameworks* de Deep Learning. Microsoft, Amazon et Facebook sont, entre autres, des acteurs de ce projet. Concrètement, il s'agit de stocker les paramètres et hyperparamètres du modèle dans un fichier, grâce à des fonctions de conversion vers ce format.

Lobe facilite la prise en main du modèle en fournissant un code complet et efficient, sous la forme d'un fichier Python contenant des fonctions de chargement du modèle, *processing* d'une nouvelle image et obtention de la prévision. Ce code est basé sur la librairie **onnxruntine**, ce qui garantit l'indépendance vis-à-vis de l'environnement d'entraînement. De plus, des dépôts GitHub vous accompagneront pour prendre en main le déploiement du modèle et proposer une application réalisant l'inférence, c'est-à-dire l'opération de prévision à partir de nouvelles données. Des exemples sont aujourd'hui disponibles pour Python avec Flask, JavaScript avec React ou sur mobile (IOS et Android). Custom Vision se démarque plutôt par la mise à disposition d'une **image Docker** déjà configurée pour être intégrée dans Azure Machine Learning.

Pour obtenir de nouvelles prévisions sans réaliser de développement, Lobe se limite au poste local alors que Custom Vision propose un **service payant d'API REST** très complet. La différence se fait ici entre un logiciel encore en version bêta et une offre plus mature.

Derrière l'entraînement « no code », une inférence « full code »

Nous sommes face à deux outils qui répondent aux cas d'usage où une adaptation très fine d'un modèle de vision par ordinateur n'est pas nécessaire. Dans un cadre généraliste, le niveau de performance est excellent, dans un temps très court et sans aucun développement ! Pour des besoins très spécifiques, il faudra mener des tests de performance et surtout suivre toutes les bonnes pratiques relatives à la constitution d'un échantillon d'images pour l'apprentissage d'un modèle.

Lobe et Custom Vision permettent de réaliser une première preuve de concept et surtout du potentiel des données à disposition, même si celles-ci sont peu nombreuses. Selon la criticité du scénario (rendre un diagnostic médical n'a pas le même impact que reconnaître une fleur !), il sera même envisageable d'utiliser les modèles obtenus dans un contexte de production.

En résumé, Lobe est très prometteur par sa simplicité d'utilisation et répond à l'approche « *no code* » pour la phase d'entraînement. Il faut toutefois attendre l'arrivée de l'offre commerciale, quand le produit sera sorti de sa phase bêta, afin d'être fixé sur sa tarification. L'inférence est aujourd'hui « *full code* », mais une partie de celui-ci est générée automatiquement. Custom Vision, de son côté, donne tout son potentiel dans une utilisation également mixte, où le portail Web se complète par l'utilisation du SDK et de l'API REST. Cette dernière, encapsulant déjà de nombreuses fonctionnalités, pourrait s'apparenter à une approche « *low code* ». Le domaine du « *serving* » (exposition de modèle prédictif) sera peut-être le prochain champ d'automatisation qui permettra d'obtenir une solution véritablement *no code* de bout en bout.

À vous de jouer !

Simplicité, la plateforme low-code “tout-en-un”

PARTIE 1

L'essor des plateformes « low-code » - un marché de 14 milliards de dollars qui a bondi de 22% en 2020 - est un fait. Même si la création d'une application est devenue à la portée de n'importe quel utilisateur « no-code » en herbe, la maîtrise du codage n'est pas encore une option s'il veut pouvoir adresser tous les besoins d'une application un peu complexe, comprenant de l'intégration à des systèmes tiers, ou des actions, des ergonomies et des règles de gestion spécifiques. Le low-code devient “la” nouvelle façon de travailler du développeur full-stack pour lui faciliter la vie sans avoir à réinventer l'informatique à chaque nouvelle technologie tous les deux ans.

François Genestin

CEO chez Simplicité Software

Tombé dans la marmite d'un Amstrad CPC 464 à l'adolescence, il est ensuite devenu ingénieur ENSTA 96 spécialisé en Système d'Information formel. Pionnier du low-code depuis 20 ans, il cofonde Simplicité Software en 2006 et en dirige la R&D. Développeur dans l'âme, il est toujours en recherche de solutions innovantes pour nous faciliter la vie.

L'avantage de modéliser son application est de décorréler par construction le langage formel de la technologie qui va l'exécuter. Une plateforme low-code est avant tout une plateforme pilotée par les modèles (on parle alors de no-code ou de model-driven) agrémentée de fonctions de fabrication, d'exécution et d'orchestration de codes/scripts spécifiques. La plateforme se charge de faire évoluer ses composants tout en garantissant une compatibilité ascendante des modèles interprétés, compilés ou exécutés à la volée.

Une promesse de “simplicité”, sans compromis

Éditeur 100 % Français, Simplicité Software édite une plateforme low-code **Simplicité** basée sur des technologies ouvertes et des langages reconnus (Java, SQL, HTML...) qui permet de développer des applications de gestion dans tous les secteurs d'activité (service public, industrie, banque, assurance, télécom…). Par exemples :

- une simple application mono-processus : le suivi d'un dossier client, une prise de commande ou d'un abonnement, la gestion de notes de frais de ses salariés,
- un référentiel de données pour tout son SI comme : la gestion d'assets, un catalogue de produits/services, la gestion des stocks, la gestion de son organisation ou de son système d'information,
- un CRM (gestion de la relation client ou autre) ou un ERP dédié (gestion d'activités particulières dans l'entreprise) pour gérer des processus stratégiques qui n'existeraient pas dans des produits du marché (type RH, comptabilité, facturation…) et qu'il faudrait compléter en s'y interfaçant.

Le dénominateur commun étant que l'application doit pouvoir se modéliser en termes de données structurées et transactionnelles (norme SQL 2016), Simplicité ne sait pas (encore) gérer des données pour des bases NoSQL ou déstructurées, car les besoins adressés sont généralement tous transactionnels sur une multitude d'objets relationnels fortement accédés et mis à jour en concurrence d'accès par des métiers (back-office) ou des usagers (front-office).

L'objectif de la plateforme n'est pas de simplifier les besoins des utilisateurs, mais de faciliter l'implémentation des fonctionnalités. Pour y parvenir, elle agrège dans un seul environnement toutes les phases du DevOps :

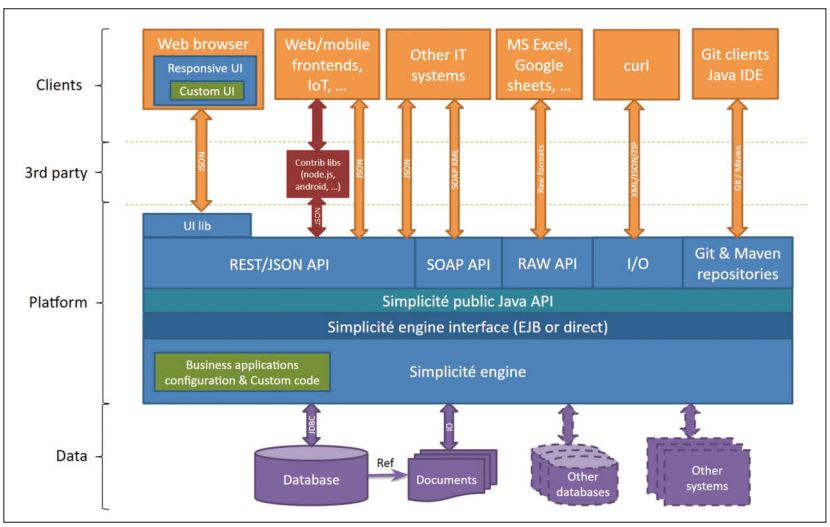
- plan : aide à la conception sous forme graphique
- build : design des fonctions par paramétrage et codage des fonctions spécifiques
- test : audit et usage/test en temps réel au fur et à mesure du design
- release/deploy : pour versionner/déployer sur environnement cloud (ou on-premise)
- operate/monitor : supervision à chaud de la plateforme, historique des usages et monitoring des performances, feedbacks

Il est également possible de surcharger la plateforme elle-même en héritant des objets natifs (User, Object, Field, Process, States, Action...) pour leur ajouter des comportements ou du code ou des libs tierces. Le runtime d'une plateforme low-code est souvent perçu comme une “boîte noire”, mais avec Simplicité il y a de nombreuses portes pour tout voir et personnaliser selon ses besoins.

L'architecture permet tout type d'intégration pour les développeurs :

- En front : Simplicité expose toute une gamme de services (REST/JSON, SOAP/XML et des helpers pour les UI développées en Node, Vue, React, Androïd…) pour accéder aux données publiques ou fortement habilitées,

Figure 1



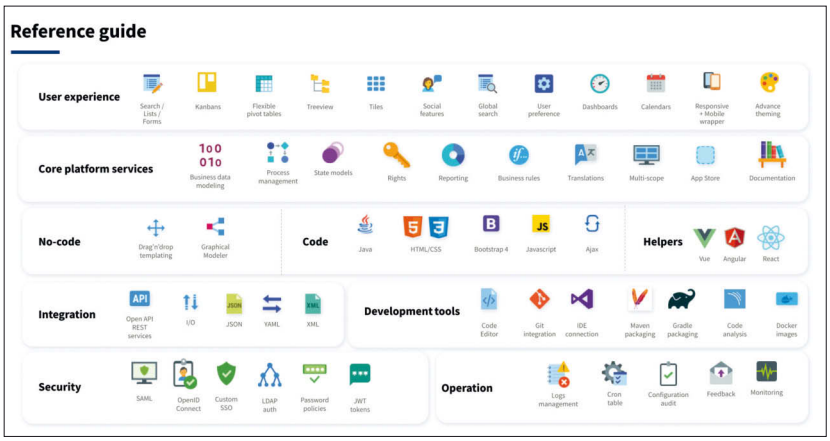


Figure 2

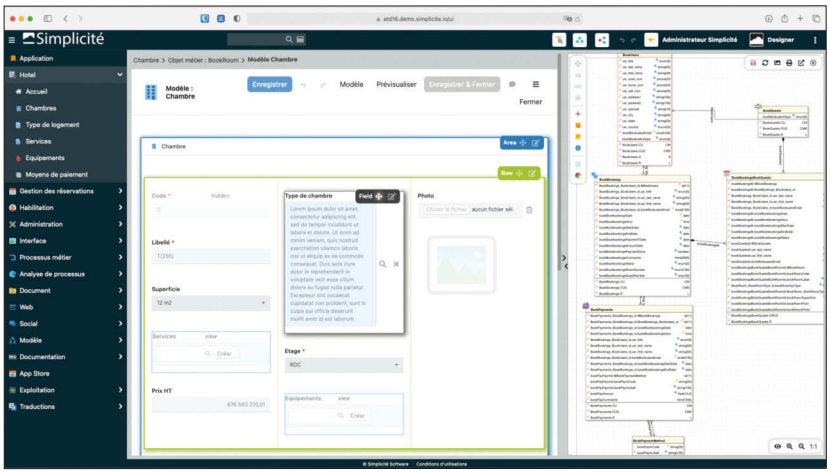


Figure 3

voir à la définition des objets eux-mêmes (les méta-data),

- En back : le runtime tourne à minima dans un container Java web/Servlet (ou Java EE/EJB) connecté à une base de données relationnelle (via JDBC/SQL) pour son fonctionnement interne. La persistance pouvant se faire également vers d'autres bases (via JDBC) ou autres systèmes (via API/webservices ou via connecteurs spécifiques par exemple Stripe, Salesforce ou DocuSign).

Figure 1

En termes de fonctionnalités, la plateforme embarque tout le nécessaire pour créer et faire évoluer une application dans le temps : **Figure 2**

Expérience utilisateur : Simplicité génère à la volée en fonction des usages le menu, la navigation, les listes simples ou groupées, les formulaires composites, les tableaux croisés dynamiques, les vues arborescentes ou de type Kanban, l'affichage en tuiles/masonry, les fonctions sociales et collaboratives, la recherche simple ou fulltext, la gestion des préférences d'affichage par utilisateur, les tableaux de bord et les graphiques, les éditions, les impressions, les interfaces responsives sur tablette et mobile, les thèmes personnalisés, les cartes géographiques, le undo/redo, etc.

Services API : Simplicité administre le modèle logique des objets et les accès physiques aux données et aux documents, la gestion des règles métier, l'orchestration des processus, l'historisation et la traçabilité des sessions et des données, la gestion des droits et des scopes, l'App Store pour diffuser ses modules et la documentation.

No-code : les citizen-users (non-développeurs) peuvent collaborer et fabriquer certaines fonctionnalités au travers d'outils visuels comme l'éditeur de formulaires et d'objets par simple glisser-déposer, et le modeleur graphique des diagrammes (objets, processus, états, droits, tables...), ou plus simplement, modifier des listes de valeurs, les aides en ligne ou les libellés des champs dans toutes les langues.

Intégration : Simplicité parle couramment l'Open API REST, Swagger, JSON, YAML et XML et intègre nativement des connecteurs vers des services tiers : Drive/JCloud, Google, Docusign, Stripe, Trello, Slack... et il est possible d'intégrer ses propres connecteurs ou de demander à l'éditeur d'en ajouter en standard s'ils peuvent intéresser tout le monde.

DevOps : Simplicité embarque son propre éditeur de code online (via ACE editor) et permet une intégration avec son IDE préféré (Eclipse, VSCode, IntelliJ ou Theia), avec un packaging projet sous Maven (ou Gradle), le repository est délivré en continue par export/import de modules (généralement sous GIT), et un audit de code assuré par SonarQube. Les images Docker sont conçues pour toute base de données relationnelles SQL-2016 (HSQLDB, PostgreSQL, MariaDB, MySQL, Oracle, SQL Server) et ouvertes au développeur (via Docker compose).

Sécurité : dans Simplicité tout accès à une donnée, un document ou un processus est fortement habilité depuis le login utilisateur, avec une intégration native de LDAP/SAML, OpenID Connect, Custom SSO, JWT tokens et même Crowd d'Atlassian. Simplicité fait l'objet de nombreux tests en termes de robustesse face aux attaques XSS/CSRF.

Opération : la plateforme supervise ce qu'elle exécute à savoir la gestion des logs, des sessions utilisateurs et des quotas, la crontab et l'historique des tâches asynchrones, des imports en masse de données, les feedback utilisateurs, le monitoring et l'analyse des performances à chaud, ou encore le clustering dans une architecture distribuée. **Figure 3**

En général, 20% du travail est réservé à l'écriture de code, règles de gestion, composants spécifiques back ou front, et 80% s'effectuent à partir de l'interface web, par simple paramétrage de formulaires, ou via le modeleur intégré et l'éditeur de template. Le but est de ne pas avoir à réinventer la roue à chaque projet et se concentrer sur le code à valeur ajoutée, sans perdre du temps à dessiner une liste ou un formulaire en relation avec la base de données, ou rendre un champ visible ou obligatoire sous condition usuelle.

L'ambition est également de permettre aux développeurs de réduire les temps de développement (pour ceux qui traînent à la cafet' ou sur le net, ou pour les bosseurs d'en faire plus dans le même temps :-), de favoriser une migration fluide vers le cloud, d'accroître l'efficacité numérique et enfin, de réduire la dette technique du code à maintenir sur des technologies mourantes à plus ou moins court terme. En clair, c'est désormais le boulot des éditeurs de plateforme low-code de garantir les montées de version techniques avec une compatibilité ascendante des modèles applicatifs. Les impacts potentiels seront limités au seul low-code spécifique : en général des API deprecated à migrer vers des plus performantes.

Une version qui fait le plein de nouveautés

La version 5 de sa plateforme s'appelle « Bonne nouvelle ». Elle améliore l'expérience de ses utilisateurs grâce à la refonte des écrans natifs de restitution, tant au niveau ergonomique UX/UI responsive que son socle technique (fin de jdk8). Son déploiement via Apple Store et Google Play a également été facilité. La gestion des utilisateurs gagne aussi en finesse en autorisant le changement de profil (Connect-as pour les testeurs et God-mode pour les administrateurs), de langues et de thèmes.

En outre, de nouveaux outils de productivité font leur apparition. Parmi eux, la gestion de la concurrence d'accès avec verrouillage ou non des objets modifiés, le multilingue centralisé, une fonction de gestion des temps (timesheet) et de nouveaux calendriers (basés sur fullcalendar.io), ou encore la gestion de tâches "libres" avec suivi d'activités et check-lists (données non structurées et librement modifiables par l'utilisateur).

La plateforme est proposée à la location en déploiement sur cloud privé, public ou hybride via Docker ou Kubernetes. Les développeurs et les designers peuvent se déployer une sandbox pour commencer à jouer, les tutos de formation sont disponibles en ligne sur le site de training https://docs2.simplicite.io.

La suite de cet article dans le prochain numéro

5 IDÉES REÇUES SUR LE LOW-CODE...

1 Le low-code ce n'est que pour des petits projets ou des petits départements IT

Faux, les plateformes low-code intègrent tout le nécessaire pour la création, le maintien et la montée en charge d'une application. Il n'y a pas de taille limite pour une application créée avec une plateforme low-code scalable comme Simplicité.

2 Le low-code, ce n'est que pour des profils non-développeurs

Faux, cela dépend des types de plateformes, on peut en distinguer plusieurs en fonction des besoins. L'analyste Forrester les distingue ainsi :

- Business Developers et Mobile for Business Developers : applications "simples" pour développeurs non professionnels, orientées UX/UI,
- Digital Process Automation : applications d'instrumentation de processus métier pour développeurs professionnels, orientées activités,
- Application Development and Delivery : applications métier complexes pour développeurs professionnels / DevOps, orientées données.

3 Les plateformes low-code sont uniquement utilisées pour la phase de développement

Faux, une plateforme low-code permet de passer très rapidement du stade de l'idée à celui d'application opérationnelle. Les plus complètes comme Simplicité intègrent tout le nécessaire pour gérer des applications dans tout le cycle projet : Concevoir > Fabriquer > Tester > Livrer > Exploiter.

4 Le low-code va rajouter du shadow IT

Faux, grâce au low-code il est possible de modéliser et centraliser tous les processus métier. De cette façon, la vieille gestion reposant sur de la bureautique, des emails, des tableurs ou des bases Access en dehors du Système d'Information (le shadow IT) peut facilement migrer vers du low-code dans le Cloud. En construisant des applications qui répondent aux besoins du métier, le designer/développeur reprend le contrôle des développements de manière industrialisée et exploitable. Le low-code est la solution pour migrer à moindre coût les usages stratégiques, mais historiquement implémentés au mauvais endroit (outils bureautiques ou technos obsolètes).

5 Avec une plateforme low-code, il n'est pas possible d'ajouter du code externe et de le débugger

Faux, la plupart des plateformes low-code intègrent un éditeur de code ou s'interfacent avec des IDE. Dans Simplicité, on peut développer des composants externes et les intégrer à la plateforme pour en étendre les fonctions natives. On peut faire du remote-debug de tout son code (spécifique et hooks) en utilisant les fonctions classiques de débogage à chaud, c'est-à-dire : mettre des points d'arrêt conditionnels, faire du pas-à-pas, modifier les objets en mémoire et faire du hot-swap des classes modifiées sans avoir à redéployer le module.

Ressources

Site internet : https://www.simplicite.fr
Forum de discussion : https://community.simplicite.io
Documentation : https://docs.simplicite.io
Site de training : https://docs2.simplicite.io
Blog : https://blog.simplicite.fr
Linkedin : https://fr.linkedin.com/company/simplicite-software
Twitter : https://twitter.com/simplicitesoftw
Facebook : https://www.facebook.com/simplicitesoftware

Vincent Thavonekham

Expert Smart Factory, Microsoft Regional Director et MVP Azure
www.factovia.fr
MUG Lyon

Maxime Billemaz

Spécialiste IoT Cloud
MUG Lyon

Digital Twins ou les Jumeaux numériques

Dans un monde où le nombre d'objets connectés est en croissance constante et où les entreprises peuvent en utiliser plusieurs milliers, il devient impératif de pouvoir modéliser, monitorer et interagir avec toutes ces entités. Le concept de « Digital Twin » est devenu notre quotidien. Nous allons explorer une solution mise à disposition par Microsoft qui permet de gérer ces interactions. Pour illustrer cela de façon concrète, nous allons considérer le Centre Georges Pompidou.

Qu'est-ce qu'un jumeau numérique

D'après la définition officielle du « Digital Twin Consortium », un jumeau numérique est "une représentation virtuelle d'entités ou de processus réels, synchronisés à une fréquence et avec une fiabilité définie". Cette notion commence à apparaître sur la « Hype Cycle for Emerging Technologies » de Gartner en 2017, derrière le Quantique. Un an après, le Digital Twin a déjà fait un bond dans une nouvelle catégorie, laissant le quantique derrière lui. La théorie est simple : il doit être possible de représenter n'importe quelle entité ou n'importe quel processus, que ce soit les différents services et les salariés de votre entreprise, votre flotte de drones ou encore les différents étages de votre immeuble avec leurs différents capteurs. Dans le bâtiment la notion similaire de BIM (Building Information Model) existe depuis longtemps ; nous allons voir comment le « Digital Twin » complète, voir même remplace le BIM.

Pour illustrer avec du code concret, nous allons choisir une implémentation du « Digital Twin » avec Microsoft **Azure Digital Twins**. En effet, Microsoft est l'un des membres fondateurs du « Digital Twin Consortium » aux côtés de DELL, Ansys, Autodesk, GE Digital, etc. C'est un écosystème rassemblant industries, universitaires et gouvernements. Son rôle est de développer et de promouvoir des technologies open source de jumeaux numériques. D'autres initiatives existent comme le BIM2TWIN soutenue par la Commission européenne.

Azure Digital twin

Avec Azure, les jumeaux numériques sont basés sur un réseau en graphes, qui permet de relier nos différents appareils entre eux et de faire des requêtes basées sur ces relations (par exemple, récupérer tous les jumeaux qui sont au troisième étage de notre bâtiment). Une des grandes forces d'Azure Digital Twins est qu'il est très facile d'utiliser de nombreuses sources pour récupérer nos données. Cela nous permet d'enrichir le modèle avec des données que nous n'aurions pas forcément dans l'objet réel, par exemple, sa position dans une usine, son « ordre de fabrication », issue d'un système ERP ou d'un serveur SQL.

Nous allons débuter le projet en modélisant notre bâtiment connecté, avec pour unique source de données celles issues de nos devices IoT **(Figure 1**). Vous noterez que cela ressemble à de la conception objet avec liens et des mécanismes d'héritages. Le tout se traduit au format JSON respectant le principe de DTDL : « Digital Definition Twin Language ». Ce JSON est un JSON-LD (JavaScript Object Notation for Linked Data), qui permet de créer des modélisa-

Figure 1
Visualisation sous forme de graphe de jumeaux numériques

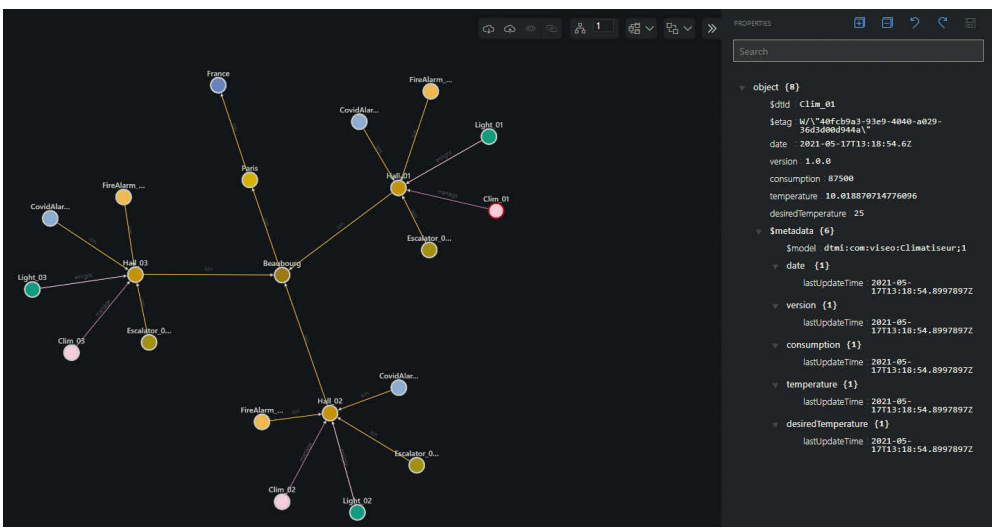


Figure 2
Modélisation DTDL d'un climatiseur (fictif)

Climatiseur.json > ...

```
{
  "@context": "dtmi:dtdl:context;2",
  "@id": "dtmi:com:viseo:Climatiseur;1",
  "@type": "Interface",
  "displayName": "Climatiseur",
  "contents": [
    {
      "@type": "Property",
      "name": "temperature",
      "schema": "double"
    },
    {
      "@type": "Property",
      "name": "desiredTemperature",
      "schema": "double"
    },
    {
      "@type": "Property",
      "name": "consumption",
      "schema": "double"
    },
    {
      "@type": "Property",
      "name": "date",
      "schema": "string"
    },
    {
      "@type": "Property",
      "name": "version",
      "schema": "string"
    },
    {
      "@type": "Relationship",
      "name": "manage",
      "target": "dtmi:com:viseo:Hall;1"
    }
  ]
}
```

tions de données tel un tableau multi-dimensionnel de clef-valeurs (**Figure 2**).

Mise en pratique : Connecter un bâtiment avec Azure

Dans les démonstrations sur Internet, nous avons souvent l'impression qu'il n'y a rien à faire et qu'il y a de la « magie » et des beaux bâtiments 3D. Nous allons voir que le métier du développeur a encore de l'avenir devant lui malgré le « No Code / Low Code » !

Connectons notre centre Georges Pompidou à son jumeau numérique en trois étapes : création de ressources Azure, mise en place de nos simulateurs de devices IoT, et récupération des données dans notre Azure Digital Twins. Le tout permettra une récupération et synchronisation bidirectionnelle des données en « near temps réel », avec de nombreux principes intégrés de façon native (protocoles d'échanges, sécurité, certificats X.509, système de « Microsoft Plug and Play », déconnexion courte ou longue d'Internet,…).

Prérequis

Pour créer nos ressources, il est nécessaire de disposer :

- D'un compte MSA (Microsoft Account) ou Office365 (ou Azure Active Directory)
- D'une souscription Microsoft Azure
- D'une invite de commande avec Azure CLI (cf. liens)
- Du code disponible sur le GitHub du MUG Lyon (cf. liens) L'intégralité du code se trouve dans le dossier « /DigitalTwin/BuildingTwin » et tous les chemins présents dans cet article sont relatifs à ce dossier.

Créer les ressources Azure

Même s'il est possible de faire tout cela via l'interface graphique d'Azure, nous préférons les lignes de commande, car ce sera plus facile à reproduire, à partager sur GitHub, ou à automatiser. Nous allons commencer par sélectionner notre souscription et créer un groupe de ressource :

```
az account set --subscription <subscription>
az group create -l francecentral -n rg-batiment-demo
```

L'IoTHub est la base de notre architecture, c'est là que vont être définis nos appareils IoT et que les données vont arriver :

```
az iot hub create --name hub-batiment-demo --resource-group rg-batiment-demo
--sku S1
az iot hub consumer-group create --hub-name hub-batiment-demo --name dtfunc
```

Nous allons ensuite créer notre ressource Azure Digital Twins :

```
az provider register --namespace 'Microsoft.DigitalTwins'
az dt create -n dt-batiment-demo -g rg-batiment-demo --assign-identity -l westeurope
```

Les Azure functions vont permettre de rediriger les données de l'IoT hub jusqu'aux jumeaux, puis des jumeaux jusqu'au SignalR qui va permettre de mettre à jour nos données instantanément dans notre application. Une fonction est forcément liée à un compte de stockage pour fonctionner correctement. Nous allons donc en créer un en premier :

```
az storage account create --name storagebatimentdemo --location francecentral --
resource-group rg-batiment-demo  --sku Standard_LRS
```

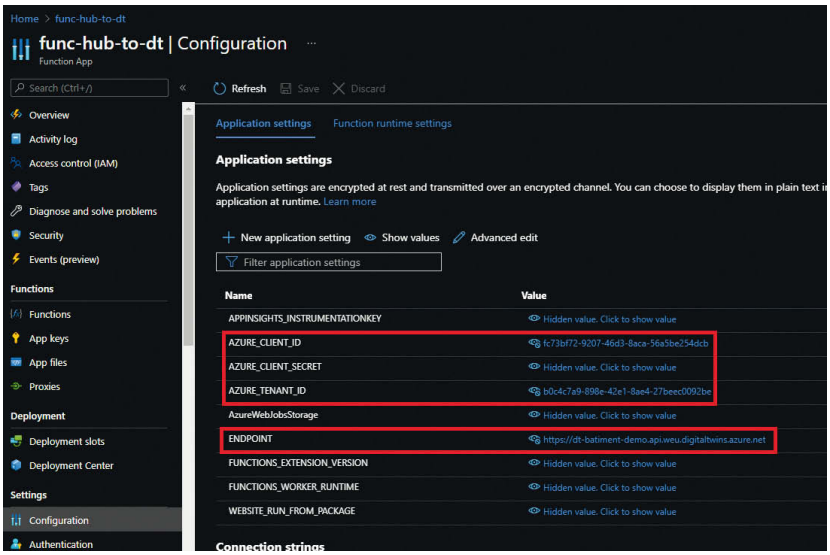


Figure 3
Configuration des variables d'environnement

```
az functionapp create --consumption-plan-location francecentral --name func-hub-to
-dt --os-type Linux --resource-group rg-batiment-demo --runtime node  -s storage
batimentdemo

az functionapp create --consumption-plan-location francecentral --name func-dt-to
-signalr --os-type Linux --resource-group rg-batiment-demo --runtime dotnet -s storage
batimentdemo
```

Les fonctions à déployer sur ces ressources sont respectivement dans les dossiers « ./AzureFuntions/DigitalTwinsFunctions » et « ./AzureFunctions/SignalRFunction » et seront présentées un peu plus loin.

Pour la fonction « func-hub-to-dt », il est nécessaire de rentrer un certain nombre de paramètres. Tout d'abord, il faut entrer la chaîne de connexion à l'event-hub qui va transmettre les données des devices iot. On peut utiliser cette commande qui va automatiquement récupérer la chaîne et la rentrer dans la configuration de notre fonction :

```
az webapp config connection-string set -n func-hub-to-dt -g
rg-batiment-demo  -t Custom
--settings eventhub_cnstr=$(az iot hub connection-string show -n
hub-batiment-demo --default-eventhub -o tsv)
```

Il est ensuite nécessaire de faire de la configuration de droits pour connecter notre fonction au Digital Twin. Nous allons créer un service principal dans votre Azure AD qui aura les droits de lire et modifier les données du Digital Twin (cf. liens). Vous obtiendrez ainsi des identifiants à rentrer dans les variables d'environnement de la fonction « func-hub-to-dt » (**Figure 3**).

Le SignalR permet de récupérer les changements de jumeaux en temps réel via un event-grid et de faire passer l'information à nos applications via un système de souscription. Créons donc le SignalR ainsi que l'event grid avant de les connecter via une route :

```
az signalr create --name sr-building-demo --resource-group rg-batiment-demo --sku
Standard_S1 --service-mode Serverless

az eventgrid topic create --name evg-batiment-demo --resource-group rg-batiment-
demo -l francecentral
```

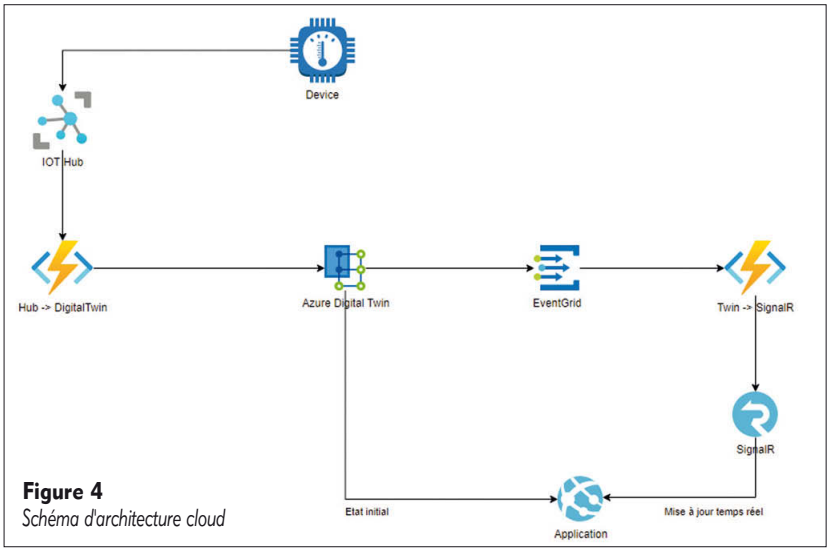


Figure 4
Schéma d'architecture cloud

Name	Type
app-batiment-visu	App Service
dt-batiment-demo	Azure Digital Twins
evg-batiment-demo	Event Grid Topic
FranceCentralLinuxDynamicPlan	App Service plan
func-dt-to-signalr	Function App
func-dt-to-signalr	Application Insights
func-hub-to-dt	Function App
func-hub-to-dt	Application Insights
hub-batiment-demo	IoT Hub
plan-batiment-visu	App Service plan
sr-building-demo	SignalR
storagebatimentdemo	Storage account

Figure 5. *Groupe de ressources Azure*

Figure 6

```
az dt endpoint create eventgrid --endpoint-name twin-endpoint --eventgrid-resource
-group rg-batiment-demo --eventgrid-topic evg-batiment-demo -n dt-batiment-demo

az dt route create -n dt-batiment-demo --endpoint-name twin-endpoint --route-
name twin-to-grid --filter "type = 'Microsoft.DigitalTwins.Twin.Update' OR type =
'Microsoft.iot.telemetry'"
```

L'architecture finale (**Figure 4**) se traduit par une liste de services Azure instanciés (**Figure 5**).
Finalement, il ne reste plus qu'à entrer nos modèles et nos instances dans Azure Digital Twins. Même s'il est possible de créer les modèles et la hiérarchie d'objets via l'interface graphique, utilisons un script node allant lire un fichier CSV pour faire un import massif (cf. GitHub, dossier « twin_creator »).
Dupliquez le fichier « env-example », renommez-le en « .env » et remplissez les variables avec les identifiants du service principal créés plus tôt. Pour initialiser l'environnement et lancer le script, utilisez « npm i » et « npm start ».
Cette application récupère la liste des modèles dans le dossier « ./assets/models » pour les importer, puis elle utilise le fichier CSV présent dans « ./assets/instances.csv » pour créer les jumeaux avec leurs données et leurs relations.

Création de devices IoT

Dans un véritable bâtiment nous avons de nombreux appareils physiques à déclarer et à connecter. L'astuce du CSV permet de gagner du temps. Dans cet article, limitons-nous à trois halls du Centre Georges Pompidou, qui contiendront 4 devices :

- Un climatiseur
- Une ampoule
- Un compteur de personnes
- Un escalator

Pour cet article, créons ces 4 devices virtuellement et agrégeons-les dans un simulateur appelé « Hall ». Cela simulera les données de tous les appareils d'un étage de notre bâtiment.
Les trois premiers devices IoT se situent dans les étages ; nous allons les paramétrer en modifiant leur « twin device », que l'on ne pas confondre avec le « Digital Twin » : Un « twin device » (**Figure 6**) est simplement un objet que l'on peut modifier depuis le cloud, là où le « Digital Twin » est tout l'objet de cet article ! Le device peut accéder à son jumeau et confirmera sa mise à jour en affichant des valeurs reportées.
Un exemple serait la température désirée pour un climatiseur, que nous avons appelée « targetTemp ». En modifiant cette valeur, le climatiseur est informé que l'on souhaite baisser ou monter la température.
Pour illustrer que l'on peut rajouter des paramètres influençant cette température, rajoutons :

- Les dimensions de la salle qui influence la température et les calculs énergétiques
- Des booléens pour allumer/éteindre les escalators et la lumière
- Un numéro de version pour la climatisation pour illustrer la gestion des mises à jour déployées à distance
- Une alerte sonore en décibels émise par l'alarme incendie

Pour créer nos trois halls qui sont considérés comme 3 devices, nous allons utiliser la commande suivante, que l'on aura à répéter pour chaque device en changeant le paramètre --device-id :

```
az iot hub device-identity create --device-id Hall_01 --hub-name hub-batiment-demo
```

Une fois les devices créés, il faut initialiser leur « twin device » avec des valeurs par défaut. N'hésitez pas à les modifier pour voir les changements dans le comportement des simulateurs :

```
az iot hub device-twin update -n hub-batiment-demo -d Hall_01 –desired '{"targetTemp":
25, "length": 10, "width": 10, "height": 10, "climVersion": "1.0.0", "isEscalatorRun":
true, "isLightRun": true, "alarmDecibel": 105}'
```

La commande suivante permet de récupérer la chaîne de connexion de notre device (à faire pour chaque device) :

```
az iot hub device-identity connection-string show --device-id Hall_01 –hub-name hub
-batiment-demo -o tsv
```

Notre simulateur (dossier « simulateurs » de GitHub) utilise une variable d'environnement DEVICES_CONNECTION_STRING qui doit être un tableau contenant nos X chaînes de connexion ([« Chaine1 », « Chaine2 », « Chaine3 »]).
Toutes les cinq secondes, les données de nos simulations sont mises à jour et envoyées dans l'IoT Hub. L'objectif est de simuler un grand nombre de devices dans chaque hall. Une fois dans le cloud, nos données doivent être transmises à travers plusieurs jumeaux numériques, et pas seulement un objet « Hall ». Pour cela, nos données vont être formatées avant d'être envoyées (**Figure 7**).
Notre simulateur crée et modifie toutes les données nécessaires à tous nos jumeaux. La fonction « formatToDT » ci-dessous crée un objet qui lui-même contient un objet pour

chaque jumeau. La clef indispensable à ces objets est le « TwinId » qui va servir à déterminer quelles données vont dans quel jumeau.
Maintenant que nos données formatées sont envoyées dans le cloud, il ne reste plus qu'à les utiliser pour modifier nos jumeaux numériques avec une Azure function.

Azure functions

Fonction HubToDigitalTwin

Les messages formatés sont maintenant dans notre fonction. Afin de la rendre la plus générique possible, cette fonction va chercher toutes les occurrences de la clef « TwinId » dans les messages entrants et créer un objet lié à la valeur de cette clef pour mettre à jour le jumeau correspondant. Pour les données qui ne sont liées à aucun TwinId, on utilise le nom du device IoT.
On peut voir ci-contre (**Figure 8**) un exemple du résultat obtenu en fonction des données d'entrées.
Une fois les données de nouveau formatées, on utilise la librairie « digital-twins-api » connectée à notre Azure Digital Twins pour mettre à jour chaque jumeau.

```
const items = extractTwins(device, message)

Object.keys(items).forEach((key) => {
  dt.updateTwin(key, items[key]).catch((err) => console.error (err))
})
```

Si les données envoyées correspondent bien aux modèles entrés dans les modèles de jumeaux, les données seront maintenant correctement mises à jour !

Fonction DigitalTwinToSignal

La fonction permettant d'envoyer en temps réel les modifications de jumeaux numériques via SignalR est une fonction basique fournie et détaillée par Microsoft (cf. liens).
Elle ne nécessite qu'un changement mineur permettant de la généraliser et d'envoyer n'importe quelle modification :
voir code complet sur programmez.com & github

Visualiser notre environnement virtuel

Nous avons maintenant des objets virtuels contenant de nombreuses propriétés liées à des objets réels. Il ne reste plus qu'à créer différents moyens, de les visualiser et interagir avec. Microsoft met à disposition une application Open Source de démonstration pour de visualiser et modifier nos jumeaux sous forme de graphe (cf. liens). Grâce à cet outil, nous pouvons visualiser un graphe représentant les connexions et la hiérarchie entre nos différents modèles, ainsi que les héritages d'objets.
Un autre graphe des jumeaux numériques présente leurs relations et leurs données. Par exemple sur le graphique (**Figure 9**), au centre se trouve le centre Beaubourg, avec 3 branches qui sont nos trois Halls. Pour le « Hall_02 », en cliquant sur « Clim_02 » nous affichons à droite les données reçues de la climatisation du hall 2, comme la température à 10,09°C et la température désirée à 45°C.
Nous avons donc la base pour illustrer les possibilités du « Azure Digital Twin », par exemple, nous allons créer de nouvelles instances, de nouvelles relations et faire des requêtes.

```
formatToDT() {
    const volume = this.twinData.length * this.twinData.height * this.twinData.width
    const globalConsumption = this.datas.lightConsumption + this.datas.escalatorConsumption + this.datas.climConsumption
    const date = new Date().toISOString()
    const isCovidAlarmRun = (this.twinData.width * this.twinData.length / this.datas.nbPeople) < 4;

    return {
        temp: {
            TwinId: "Clim_" + this.deviceNumber,
            date,
            version: this.twinData.climVersion,
            consumption: this.datas.climConsumption,
            temperature: this.datas.temp,
            desiredTemperature: this.twinData.targetTemp
        },
        escalator: {
            TwinId: "Escalator_" + this.deviceNumber,
            date,
            isRunning: this.twinData.isEscalatorRun,
            consumption: this.datas.escalatorConsumption
        },
        light: {
            TwinId: "Light_" + this.deviceNumber,
            date,
            isRunning: this.twinData.isLightRun,
            consumption: this.datas.lightConsumption,
            luminosity: this.datas.luminosity
        },
        fireAlarm: {
            TwinId: "FireAlarm_" + this.deviceNumber,
            date,
            status: this.datas.temp > 80 ? "ON" : "OFF",
            sounds: this.twinData.alarmDecibels
        },
        covidAlarm: {
            TwinId: "CovidAlarm_" + this.deviceNumber,
            date,
            status: isCovidAlarmRun ? "ON" : "OFF",
            printedText: isCovidAlarmRun ?  `Too many people in the Hall : ${this.datas.nbPeople}` : `${this.datas.nbPeople} people in the Hall : Good`
        },
        date,
        width: this.twinData.width,
        height: this.twinData.height,
        length: this.twinData.length,
        eco: volume * globalConsumption / 1000,
        globalConsumption,
        nbPeople: this.datas.nbPeople
    }
}
```

Figure 7. *Données formatées pour l'envoi dans le cloud*

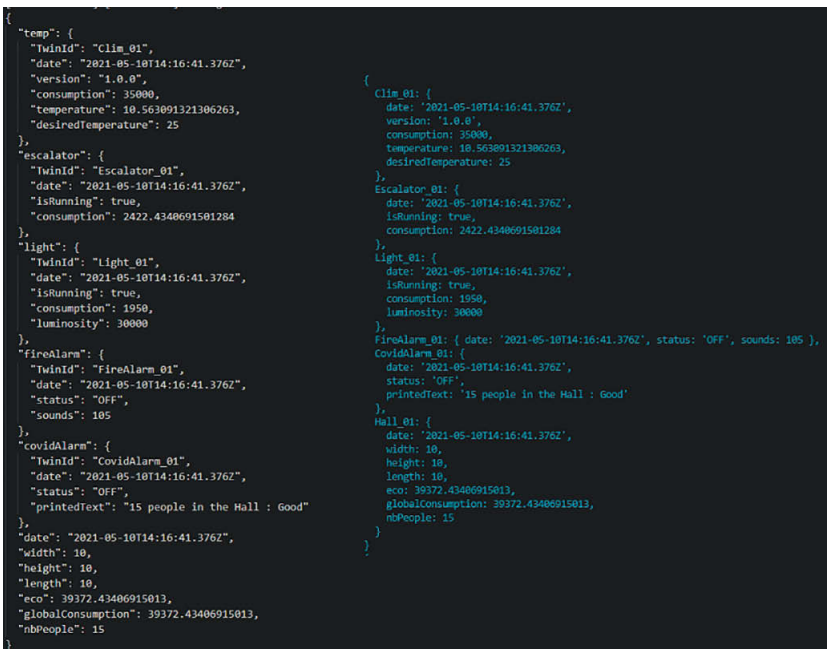


Figure 8
Création des différents objets pour mise à jour des jumeaux

Considérons le cas où il y a trop de monde présent dans le hall et qu'à cause du COVID19, nous soyons obligés de lancer une alarme. Il est alors facile de réaliser une requête pour voir toutes les alarmes covid qui sont actuellement déclenchées, car :

```
select alarm from digitaltwins alarm where alarm.$metadata.$model='dtmi:com:viseo:
CovidAlarm;1' and alarm.status = 'ON'
```

Par défaut dans cette application, les données ne se mettent pas à jour en temps réel, elles sont juste récupérées quand on clique sur un nœud. Pour avoir des informations temps réelles, nous allons créer une application Azure web App et la connecter à notre SignalR.
Une fois les premiers tests effectués, nous pouvons commencer à créer des applications plus intéressantes et visuelles.
Ci-dessous, deux exemples.
Le premier est une simple application web react qui peut être trouvée dans le dossier « site_web ». Le serveur node peut :

- Se connecter à un SignalR pour avoir les données en temps réel

```
const endpoint = process.env.SIGNALR_ENDPOINT;

if (endpoint) {
  const hubConnection = new signalR.HubConnectionBuilder()
    .withUrl(endpoint)
    .build()

  hubConnection.start()
    .then(() => console.log('SignalR Started'))
    .catch(err => console.log('Error connecting SignalR - ' + err));

  hubConnection.on('newMessage', (message) => {
    io.emit('twinData', message)

  });
}
```

- Demander directement à Azure Digital Twin les données toutes les secondes.

```
const datas: any[] = [];

const forbiddenKeys = ["$metadata", "$etag"]
for (const twinId of this.twinIdList) {
  const data = await this.service.getDigitalTwin(twinId)
  // datas.push(data.body)

  const filteredData = Object.keys(data.body)
    .filter(key => !forbiddenKeys.includes(key))
    .reduce((obj, key) => {
      obj[key] = data.body[key];
      return obj;
    }, {});

  datas.push(filteredData)
}
this.sendToSubscriber(this.formatToFront(datas))
```

Une websocket permet de passer les données à la partie front-end. On peut ainsi faire un affichage simple (**Figure 10**) ou créer des popups d'alertes en semi-temps réel.

Comme nous l'avons mentionné plus tôt, nos devices simulés possèdent un « device twin » qui permet de leur transmettre des informations comme la température ciblée. Il est donc facile depuis notre application web de créer un formulaire de modification (**Figure 11**). En validant ce formulaire, le « twin device » est modifié. Le device responsable de la simulation est alerté et récupère les nouvelles données, qui vont influencer son comportement.

Application avec de la 3D et réalité virtuelle/augmentée

Au-delà des simples interfaces web précédentes, les possibilités de visualisations sont nombreuses comme ce que l'on peut retrouver sur Internet. Par exemple, illustrons les possibilités d'ouverture et d'intégration de la technologie issue de ce consortium Open Source avec une représentation 3D. L'intégration d'Azure Digital Twins dans une modélisation 3D se fait par exemple via Unity3D (d'autres exemples notamment avec de la robotique se trouvent dans le repo GitHub du MUG Lyon).

Créons un petit projet avec Unity avec de simples formes (**Figure 12**) : la barre oblique jaune symbolise l'escalator, les 2 boules jaunes représentent 2 lumières allumées, la pilule verte représente l'alarme COVID-19. Nous avons également les écrans qui restituent du texte en near temps-réel.

Le but est de faire changer les couleurs immédiatement en fonction des données précédentes qui arrivent : l'alarme incendie en vert si tout va bien. Nous avons également placé des tooltips dynamiques affichant le détail des données, lors du survol de la souris sur chaque élément.

Une synthèse de l'ensemble est disponible en vidéo (cf. liens).

Conclusion

Le sujet est vaste, et nous n'avons couvert qu'une infime partie ; il reste la continuité numérique avec la partie CAD (Computer Aided Design) où Autodesk figure parmi les piliers du Digital Twin Consutium, les PLM (Product Lifecycle Management), le principe de Microsoft Plug and Play, ou les extensions des Twin devices, par exemple via un projet en

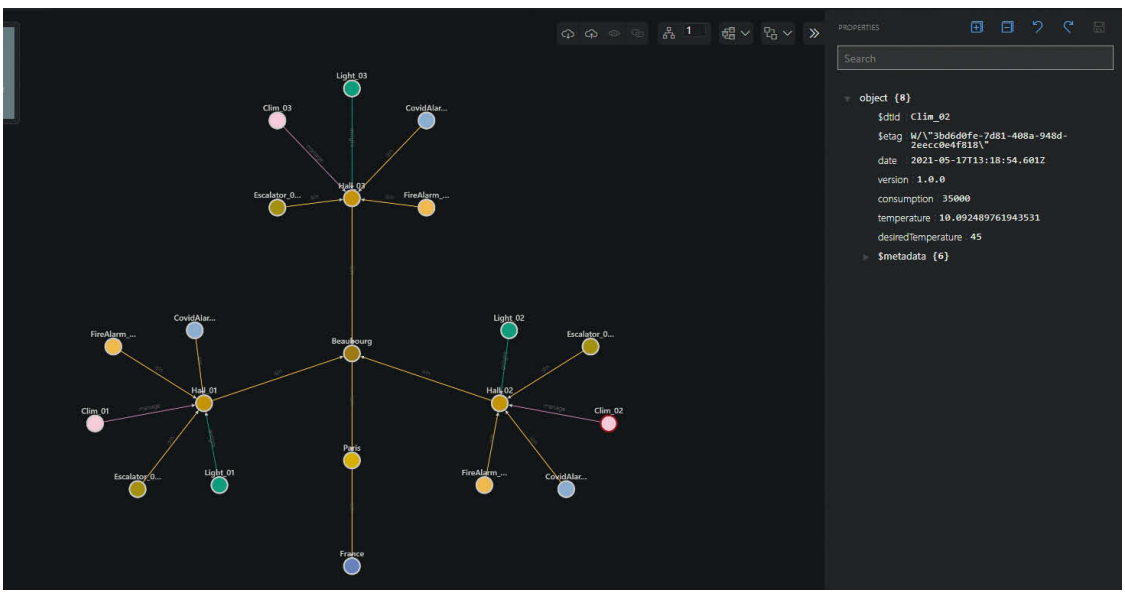


Figure 9
Graphe de visualisation des instances de jumeaux

R&D pour faire des simulations physiques (**Figure 13**.). Et ce sera l'occasion de partager notre expérience dans d'autres articles ! Et voilà pour la découverte d'Azure Digital Twin. N'hésitez pas à essayer de votre côté ou à aller découvrir d'autres technologies de jumeaux numériques et de nous partager cela au MUG Lyon !

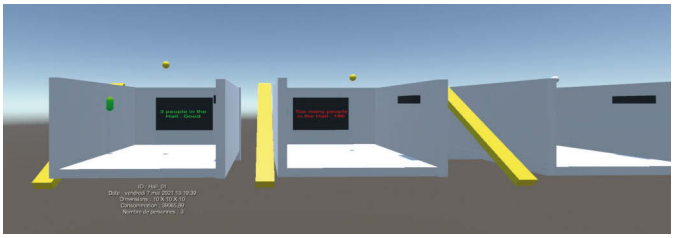


Figure 12
Modélisation 3D Unity

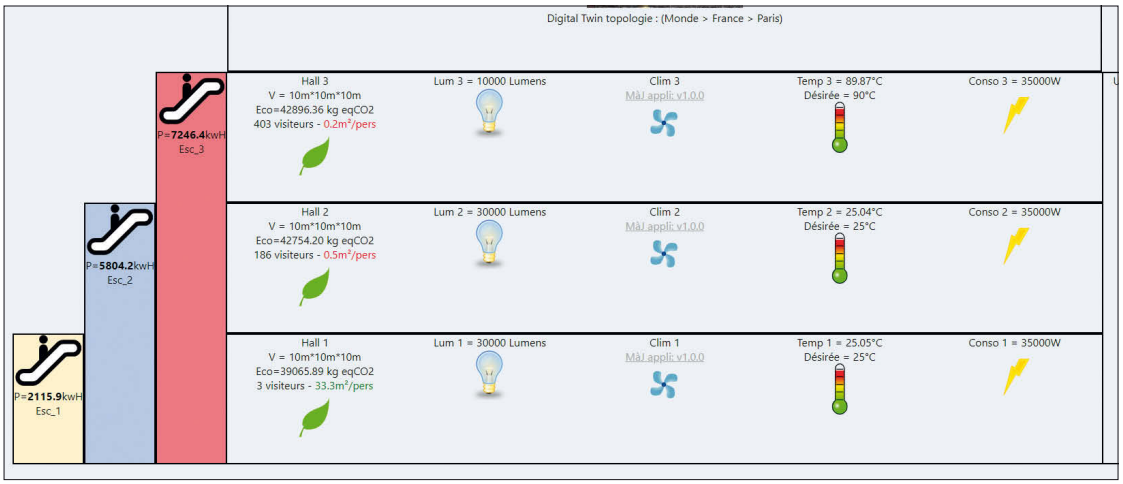


Figure 10. *Application web avec visualisation temps-réel*

Figure 11
Formulaire de modification du twin device

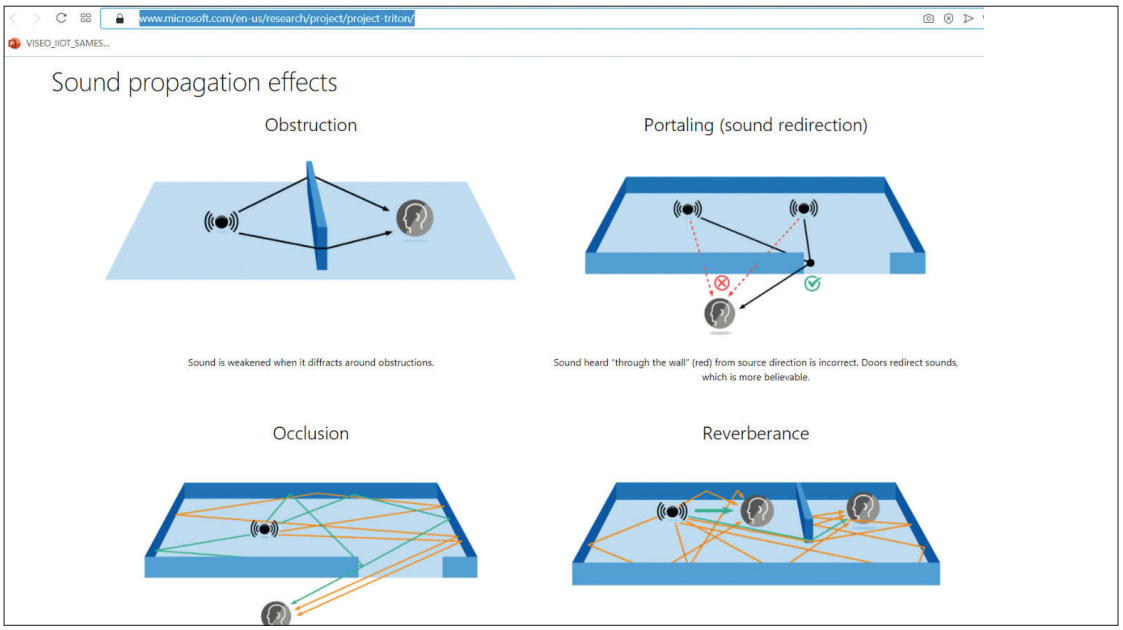


Figure 13. *Projet de R&D Triton pour compléter DT via des simulations de propagation de sons*

Liens pour aller plus loin :

Digital Twin Consortium https://www.digitaltwinconsortium.org/cgi-bin/dtcmembersearch.cgi
Azure CLI : https://docs.microsoft.com/en-us/cli/azure/install-azure-cli
Github : https://github.com/muglyon/SmartFactory/tree/master/DigitalTwin
Créer un service principal dans Azure AD : https://docs.microsoft.com/en-us/azure/digital-twins/how-to-create-app-registration
Données temps réel des jumeaux numériques via SignalR : https://docs.microsoft.com/en-us/azure/digital-twins/how-to-integrate-azure-signalr
Visualiser et modifier DT : https://github.com/Azure-Samples/digital-twins-explorer
Synthèse en vidéo : https://youtu.be/F6-8DXnpLJk
Intégration de DT par Bentley System : https://www.bentley.com/en/products/product-line/digital-twins/itwin et https://customers.microsoft.com/fr-fr/story/806028-bentley-systems-partner-professional-services-azure
Recherches en cours BIM2TWIN : https://cordis.europa.eu/project/id/958398
Projet de R&D Triton : https://www.microsoft.com/en-us/research/project/project-triton/
Microsoft Plug and Play : https://docs.microsoft.com/en-us/azure/iot-pnp/overview-iot-plug-and-play

Denis Duplan
Sociologue et développeur à ses heures.
Blog :
http://www.stashofcode.fr

Boucle d'événements et multithreading dans Node.js

En une dizaine d'années, Node.js s'est imposé comme un élément central des architectures Web et de l'univers JS. À lire les articles inventoriant les grandes entreprises qui l'utilisent, Node.js est utilisé par Netflix, eBay, PayPal, et j'en passe.

Joli succès pour cette technologie sortie de l'esprit de Ryan Dahl, qui a su repenser « *out of the box* » une manière de fonctionner des serveurs HTTP que plus personne ou presque n'interrogeait, se reposant dessus comme sur un acquis.
Pourtant, en dépit de sa popularité, il apparaît que les bases du fonctionnement de Node.js sont très mal maîtrisées. Au cœur du sujet : le fonctionnement de la boucle d'événements, et la place qu'elle laisse au multithreading.

Avertissements

Dans ce qui suit, je vais évidemment me baser sur une présentation de Node.js par Ryan Dahl, qui en est l'auteur, mais aussi sur des présentations de deux contributeurs à Node.js à l'occasion d'une conférence consacrée à la technologie fin 2016 : Bert Belder et Sam Roberts.
Ces présentations sont anciennes, mais l'on verra qu'elles restent importantes, et que cette importance ne se réduit pas à leur contenu : elle s'étend à leur existence même. Autrement dit, ce qu'il sera intéressant de relever, ce n'est pas seulement ce qui est expliqué, c'est aussi pourquoi il faut l'expliquer.
Je ne rentrerai pas totalement dans le détail de la boucle d'événements et de la place du multithreading. Mon ambition est de dégrossir des sujets qui me sont apparu très grossièrement, pour ne pas dire très vulgairement présentés dans de nombreux articles sur le Web, lorsque j'ai cherché à en savoir plus.

Node.js, la version simple (voire simpliste ?)

La nature de Node.js est un peu délicate à saisir. Node.js doit être vu comme un programme qui exécute notamment un moteur, lequel s'appuie sur l'interpréteur de JavaScript tiré de Chrome, le moteur V8, pour exécuter des programmes JavaScript. Par ailleurs, Node.js donne à ces programmes accès à une API extensible, qui leur permet d'attaquer le système d'exploitation plutôt que le navigateur - vu d'un programme, l'objet global principal est process, et non window.
Basiquement, exécuter un script en ligne de commandes à partir de la console…

```
C:\Temp\node-v14.15.4-win-x64> node
Welcome to Node.js v14.15.4.
Type ".help" for more information.
> i=1
1
```

…et au-delà, exécuter un script depuis un fichier :

```
C:\Temp\node-v14.15.4-win-x64> node test.js
```

Pour installer Node.js, il suffit de récupérer https://nodejs.org/en/download la version pour Windows 64 bits au format ZIP, et de la décompresser dans un dossier.
Node.js est accompagné de npm, acronyme pour Node.js Package Manager. Pour un programmeur en Python, c'est l'équivalent de pip (Python Installer Program). Bref, c'est un outil en ligne de commandes qui permet de gérer les packages auxquelles les programmes peuvent accéder, en permettant d'abord d'en installer parmi ceux qui figurent sur le dépôt PyPI (Python Package Index).
Un package intégré à Node.js est « http ». Il permet de faire tourner un serveur HTTP. Par exemple, un fichier server.js :

```
var http = require ('http');
var server = http.createServer (function (request, response) {
    response.write ('Hello world');
    response.end ();
});
server.listen (3000);
```

Donc tout simplement lancer le serveur en ligne de commandes...

```
C:\Temp\node-v14.15.4-win-x64> node server.js
```

…et y accéder via un navigateur :

```
http://localhost:3000
```

La particularité d'un tel serveur est une particularité de Node.js. Pour le comprendre, rien de mieux que de reprendre un exemple donné dans une vidéo « *An Introduction to Node.js with Ryan Dahl* », (1). Dans cette vidéo, l'auteur de Node.js compare deux manières dont on pourrait écrire un programme qui doit commander l'exécution d'une fonction après deux secondes.
En PHP, il faudra demander au moteur de se bloquer deux secondes avant d'exécuter la fonction :

```
sleep (2);
echo ("Hello world");
```

Avec Node.js, il faudra demander au moteur d'exécuter la fonction après deux secondes, en lui fournissant sous la forme d'une callback :

```
setTimeout (function () { console.log ("Hello world") }, 2000);
```

Node.js a été conçu pour offrir aucune possibilité pour un programme de bloquer le moteur, c'est-à-dire condamner le moteur à ne rien faire. Ce qui permet à Ryan Dahl de dire, au prix d'un abus de langage où il assimile Node.js au moteur, que « *Node.js doesn't sleep* ».

Node.js, la version complexe (voire compliquée ?)

À vrai dire, cette formule est assez malheureuse, car elle réduit Node.js à sa boucle d'événements, et laisse donc entendre que Node.js ne fonctionnerait jamais qu'avec un seul thread.

Or, il faut souligner que le multithreading dans Node.js apparaît aussi mal compris que le fonctionnement de sa boucle d'événements, deux sujets liés. Il suffit de faire quelques recherches sur ces sujets sur le Web pour s'en convaincre. Plus généralement, c'est ce que pointe Sam Roberts, au début d'une présentation qu'il consacre à la boucle (2). Pour justifier l'intérêt du sujet – pourquoi revenir sur un fonctionnement de base devant un public intéressé par les usages avancés ? –, il rapporte que chaque fois qu'il interroge des personnes sur le fonctionnement interne de Node.js :

« one of my go-to questions is what's a Node event loop and my secondary question is Node single-threaded or multi-threaded, and a suprisingly number of people had an either unclear, wrong or muddy answer to those questions. »

C'est une nouvelle illustration d'un problème toujours plus commun : depuis que le Web a doté chacun d'un porte-voix, il est devenu très difficile de trouver une explication valable du fonctionnement interne nécessairement complexe d'une technologie simple à utiliser.

Ainsi, on entend souvent dire que Node.js est single-threaded. C'est un abus de langage, qui découle d'une réduction de Node.js au moteur, lui-même souvent réduit à l'interpréteur, du fait que v8 est désigné comme un moteur de JavaScript. Prétendre que Node.js est single-threaded, ce n'est vrai que d'un certain point de vue, celui des programmes que Node.js exécute, car le moteur est une boucle d'événements qui s'exécute dans un unique thread. Pour l'OS qui exécute Node.js, ce dernier est bel et bien multihreaded.

Quant à la boucle d'événements, Bert Belder, contributeur à Node.js, montre dans une autre présentation (3) qu'elle fait autant l'objet de conceptions erronées, en démontant les schémas incongrus de la boucle qui sont légion sur le Web.

De quoi en retourne-t-il vraiment ? Bert Belder explique que pour un script donné, la boucle répète la séquence suivante :

- passer en revue une liste des timers programmés par setTimeout ()ou setInterval (), et appeler les callbacks de ceux qui ont expiré ;
- appeler une fonction « unicorn », qui, durant un temps limité, vérifie si des opérations d'I/O demandées au système sont terminées et retourne les événements correspondants s'il y en a ;
- exécuter le code dont l'exécution doit être immédiate, demandée par setImmediate () ;
- nettoyer, c'est-à-dire déterminer si plus aucun timer ni opération I/O n'est en cours, et de là acter la fin du script donc la sortie de la boucle.

Bref, comme défini par Sam Roberts, la boucle des événements de Node.js, c'est *« a semi-infinite loop, polling and blocking on the O/S until some in a set of file descriptors are ready »* – il évoque des descripteurs, car il se fonde sur l'implémentation de Node.js pour Linux. La boucle se termine quand elle n'a plus rien à attendre, ce qu'elle apprécie par l'absence de références sur des ressources qui induisent une attente, comme un timeout, une socket… Depuis un programme, il est possible d'appeler une méthode .unref () pour signifier qu'il n'est plus utile d'attendre une telle ressource.

Un point que Bert Belder évoque un peu rapidement au regard de son caractère essentiel, c'est que pour permettre la montée en charge, les auteurs de Node.js font tout leur possible pour éviter d'utiliser des threads lorsqu'il faut utiliser des fonctions du système d'exploitation, au point de ne pas hésiter à passer par le kernel plutôt que par les bibliothèques – ce qui n'est malgré tout pas une garantie de succès.

On comprend que dans un monde idéal, la boucle devrait se contenter de consulter le système pour savoir si les opérations qu'elle demande à ce dernier d'accomplir ont été réalisées, plutôt que le système ne vienne l'interrompre - toute la différence entre polling et interruption, bien familière pour ceux qui ont programmé bas niveau, comme en assembleur sur Amiga. Toutefois, ce n'est pas toujours possible.

Pourquoi les auteurs de Node.js souhaitent-ils cela ? Parce que c'est ainsi que la boucle peut maîtriser la séquence des opérations qu'elle accomplit, et donc le temps qu'elle y consacre. C'est une condition essentielle pour optimiser le traitement des événements, donc les performances de Node.js généralement.

Dès lors, si la boucle doit malgré tout demander au système d'accomplir une opération durant laquelle ce dernier doit l'interrompre pour l'informer du déroulement de l'opération en question, la boucle crée un thread qui sera interrompu à sa place. Le thread stocke une trace de cette interruption à un endroit où la boucle peut le trouver quand elle souhaite. Autrement dit, le thread joue les intermédiaires pour transformer une mécanique à base d'interruption en un mécanisme à base de polling au moyen d'un buffer. Eh oui, cela s'apparente clairement au pipe.

C'est clairement embarrassant pour les auteurs de Node.js, comme l'analyse Bert Belder :

« So if you need to make any diagrams, please don't pretend everything happens in a thread pool. That's very painful for me, because we try very hard not to do that. »

En effet, mettre en place le machin évoqué à l'instant ne va pas sans consommer des ressources, et donc limiter la capacité à monter en charge. Ce n'est donc jamais que forcé et contraint par le système d'exploitation que les auteurs de Node.js se résignent à utiliser des threads. D'où l'existence d'un petit pool de quatre threads malgré tout, nombre qu'il est possible d'ajuster via la variable d'environnement UV_THREADPOOL_SIZE.

Quelles opérations passent par un thread ?

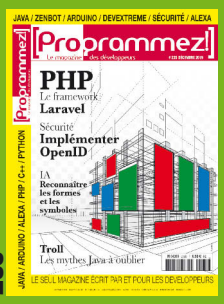
Sam Roberts évoque fin 2016 les opérations :

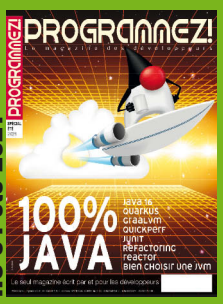
- fs.* () ;
- dns.lookup (), car elle fait un appel asynchrone à getaddrinfo () ;
- crypto.randomBytes () et crypto.pbkdf2 () ;
- http.get () et http.request () si cela implique la résolution d'un nom de domaine.

Au passage, il convient de souligner l'importance d'un autre point que Bert Belder évoque aussi un peu rapidement alors qu'il est essentiel : l'implémentation de Node.js dépend du

Les anciens numéros de PROGRAMMEZ!

Le magazine des développeurs





Tarif unitaire 6,5 € (frais postaux inclus)

TECHNOSAURES

Le magazine à remonter le temps !

Prix unitaire : **7,66 €** (frais postaux inclus)

N°4 Standard **10 €**

N°4 Deluxe **15 €**

☐ 234 : ___ ex ☐ 241 : ___ ex
☐ 235 : ___ ex ☐ HS1 été 2020 : ___ ex
☐ 236 : ___ ex ☐ 242 : ___ ex
☐ 238 : ___ ex ☐ 246 : ___ ex
☐ 239 : ___ ex ☐ 247 : ___ ex
☐ 240 : ___ ex ☐ HS4 été 2021 : ___ ex

soit ___ exemplaires x 6,50 € = ___ €

Technosaures ☐ N°1 ☐ N°2 ☐ N°3 ☐ N°5 ☐ N°6
soit ___ exemplaires x 7,66 € = ___ €
☐ N°4 Deluxe ..... 15 €
☐ N°4 Standard ..... 10 €

..... ___ €

Commande à envoyer à :
Programmez!
57 rue de Gisors
95300 Pontoise

soit au **TOTAL** = ___ €

☐ M. ☐ Mme ☐ Mlle Entreprise : ___ Fonction : ___

Prénom : ___ Nom : ___

Adresse : ___

Code postal : ___ Ville : ___

Règlement par chèque à l'ordre de Programmez ! I Disponible sur www.programmez.com

Offres pouvant s'arrêter à tout moment, sans préavis PROG 248

système d'exploitation. Linux et Windows n'ont pas les mêmes bibliothèques ni le même kernel. Oui, mais cela implique autre chose, à savoir que la possibilité d'échapper au threading pour réaliser une opération peut varier selon le système d'exploitation. En conséquence, il faut souligner l'importance de cette remarque de Sam Roberts, qui base sa présentation sur l'implémentation de Node.js pour Linux :

« *It's also very difficult, if you are not really familiar with the underlying system calls, right now it's not documented in the Node API, how things are implemented, and it will matter to you eventually.* »

Autrement dit, pour vraiment optimiser l'utilisation de Node.js sur un système d'exploitation donné, il faudrait étudier le code de Node.js sur l'OS en question.

Pour conclure

Terminons en reprenant les exemples donnés par Ryan Dahl dans sa présentation déjà mentionnée. Node.js interdisant à un programme de bloquer le moteur : il est clair que ce dernier peut exécuter plusieurs programmes « en parallèle ». Ce ne sont jamais que des événements qui se rajoutent dans la file. Toutefois, cela a un coût pour le développeur, car il doit structurer son code sous la forme d'imbrications de callbacks, ce qui débouche assez rapidement sur le fameux « *callback hell* ». Or il se trouve que cette écriture a été considérablement facilitée avec l'apparition de l'objet Promise dan ECMAScript 2015, puis des instructions async et await dans ECMAScript 2017, innovations reprises dans JavaScript, fonctionnalités détaillées longuement par votre serviteur dans un long article publié dans votre magazine préféré.

On comprend que si Node.js est utilisé sur une machine pour exécuter des scripts à la demande d'un serveur HTTP sur requête de clients, ce serveur va pouvoir traiter beaucoup plus de requêtes qu'un serveur classique, qui crée un thread par requête.

Comme par définition l'exécution d'un script n'est jamais bloquante, le serveur n'est pas contraint de créer un thread par requête pour éviter d'être bloqué par le script exécuté pour traiter cette dernière. Partant, le serveur n'est pas limité par le nombre de threads que les ressources qui lui sont allouées lui permettent de créer. Et donc il ne se trouve jamais contraint de refuser le traitement d'une requête alors qu'il ne fait rien, pour la mauvaise raison qu'il ne peut plus créer de threads sur l'instant, alors même que certains de ses threads sont occupés à ne rien faire dans l'attente de la fin d'une opération quelconque - expiration d'un délai, écriture sur le disque, etc.

Un intérêt du package « http » est qu'il fonctionne en HTTP/1.1. Cela permet notamment au serveur de consommer encore moins de ressources, puisqu'il n'a pas à stocker le résultat d'une requête pour la renvoyer en une fois : il peut la renvoyer par morceaux sur une connexion persistante.

Dans ces conditions, une réponse contiendra les en-têtes suivants :

```
Connection: keep-alive
Transfer-Encoding: chunked
```

Très pratique si la réponse est volumineuse et peut être constituée progressivement, comme les enregistrements d'une table extraits par bloc d'une base de données.

Au passage, rappelons qu'une connexion persistante est une connexion TCP maintenue d'une requête à une autre, si bien que le client et le serveur peuvent s'épargner d'avoir à négocier l'ouverture d'une connexion : cela permet d'éviter de consommer des ressources, de perdre du temps, et d'engorger le réseau avec des paquets relatifs à l'ouverture. Aussi, cela permet de pipeliner les requêtes et les réponses, le fait qu'elles circulent via une seule et même connexion permettant aux parties d'en reconnaître l'ordre – la spécification HTTP/1.1 impose que sur une connexion persistante, le serveur renvoie des réponses dans l'ordre dans lequel il a reçu des requêtes.

La lecture de la spécification de HTTP/1.1 révèle qu'une connexion est persistante par défaut depuis cette version du protocole. L'en-tête est un héritage de HTTP/1.0, où ce n'était pas le cas, et où le mécanisme pour établir une connexion persistante était différent et posait un problème au niveau du proxy. Pour la rétrocompatibilité, la spécification indique qu'il ne faut pas établir de connexion persistante HTTP/1.1 avec un client qui demande une connexion persistante HTTP/1.0. Bref, si le client et le serveur communiquent directement en HTTP/1.1, n'est-il pas possible de se passer de cet en-tête ? À voir…

Un autre package, « net », permet de créer une socket TCP. Par exemple, un fichier socket.js :

```
var net = require ('net');
var server = net.createServer (function (socket) {
    socket.write ('Hello world\n');
    socket.on ('data', function (data) {
        socket.write ('You sent: ' + data + '\n');
    });
});
server.listen (3000);
```

Donc, tout simplement lancer le serveur en ligne de commandes… :

```
C:\Temp\node-v14.15.4-win-x64> node socket.js
```

…et y accéder via l'outil ncat de nmap :

```
C:\Temp\nmap-7.80> ncat.exe localhost 3000
```

Références

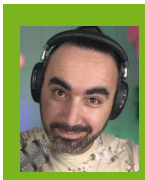
(1) An Introduction to Node.js with Ryan Dahl (https://www.youtube.com/watch?v=jo_B4LTHi3I)

(2) Node's Event Loop From the Inside Out by Sam Roberts, IBM (https://www.youtube.com/watch?v=P9csgxBgaZ8)

(3) Morning Keynote- Everything You Need to Know About Node.js Event Loop by Bert Belder, IBM (https://www.youtube.com/watch?v=PNa9OMajw9w)

Bonnes pratiques Angular

Dans ce dossier consacré aux bonnes pratiques de développement Angular, nous aborderons certains points qui nous semblent être importants lors du développement d'applications Web avec Angular. Il est important de noter que ces préconisations sont basées sur notre propre expérience en tant qu'à la fois développeur d'applications et aussi contributeur au projet Angular. Votre expérience est probablement différente et votre point de vue serait probablement également différent.

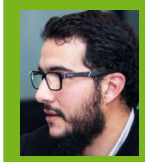


Wassim Chegham

Senior Developer Advocate chez Microsoft

Core contributeur à Angular et Node.js

GDE pour Google



Chihab Otmani

JavaScript Tech Lead Indépendant

Formateur et Community Organizer

Performance

Vous avez probablement lu ou entendu cette citation : "*Make it work, make it right, make it fast*" de Kent Beck. Nous allons appliquer ce conseil à cette section consacrée aux performances et optimisation de nos apps Angular. Avant de vouloir appliquer ces préconisations, assurez-vous que votre application Angular a réellement besoin d'être optimisée. Dans la plupart des cas, cela ne sera pas nécessaire.

JIT vs AOT

Avant de déployer nos applications Angular en production, nous devons passer par une phase de build. Pour faire simple, cette phase permet de convertir le code Angular (TypeScript et templates HTML) en un ensemble de fichiers JavaScript appelés *bundles* qui seront directement interprétés par le navigateur.

Pour être plus précis, Angular passe par une phase de compilation permettant de transformer le code TypeScript ainsi que les templates HTML en un code JavaScript optimisé pour les moteurs JavaScript, notamment V8. **Figure 1**

Jusqu'à la version 11, Angular offrait deux modes de compilations : Just-In-Time (JIT) et Ahead-of-Time (AOT(1)). Ces deux modes de compilation sont assez répandus en ingénierie logicielle. **Figures 2 et 3**

Lors de la compilation JIT, le code Angular est compilé à la volée directement au runtime, dans le navigateur. La compilation AOT quant à elle permet de compiler le code Angular pendant la phase de build produisant un code JavaScript beaucoup plus optimisé en appliquant des techniques de compilation telles que le *tree shaking* ou encore le *constant folding*. Le code JavaScript produit est plus léger et donc rapide à se lancer, comme le montre la **Figure 4.**

```
{
 "projects": {
  "my-existing-project": {
   "architect": {
    "build": {
     "options": {
      ...
      "aot": true,
     }
    }
   }
  }
 }
}
```

À noter qu'il peut arriver que vous ayez des erreurs TypeScript lors de la compilation en mode AOT, car ce mode est beaucoup plus strict. Ceci est d'autant plus important à signaler, car depuis la v12(2), le mode strict de TypeScript a été activé par défaut. Le but étant de permettre la détection d'erreurs le plus tôt possible en phase de développement. En savoir plus sur le mode strict(3).

Lazy loading ou code-splitting des routes

Par défaut, les modules Angular sont automatiquement chargés et interprétés au lancement de l'application. Cependant, il n'est souvent pas nécessaire de charger l'intégralité de ces modules immédiatement, surtout si votre application est assez conséquente. Cela risquerait de ralentir le lancement de l'application et dégrader le *Time-To-Interactive* (TTI).

Le TTI est une métrique qui mesure le temps que met une page Web à s'afficher avant que l'utilisateur puisse interagir avec – avant qu'elle devienne "interactive". Plus la valeur de votre TTI est élevée, plus l'utilisateur va devoir attendre. En

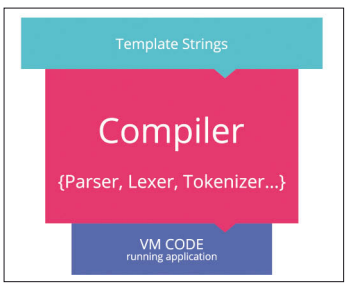
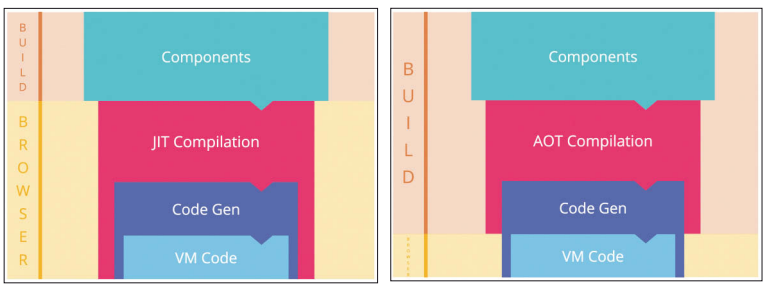


Figure 1



Figures 2 et 3

(1) *https://slides.com/wassimchegham/demystifying-ahead-of-time-compilation-in-angular-2-aot-jit*

(2) *https://blog.angular.io/angular-v12-is-now-available-32ed51fbfd49*

(3) *https://angular.io/guide/strict-mode*

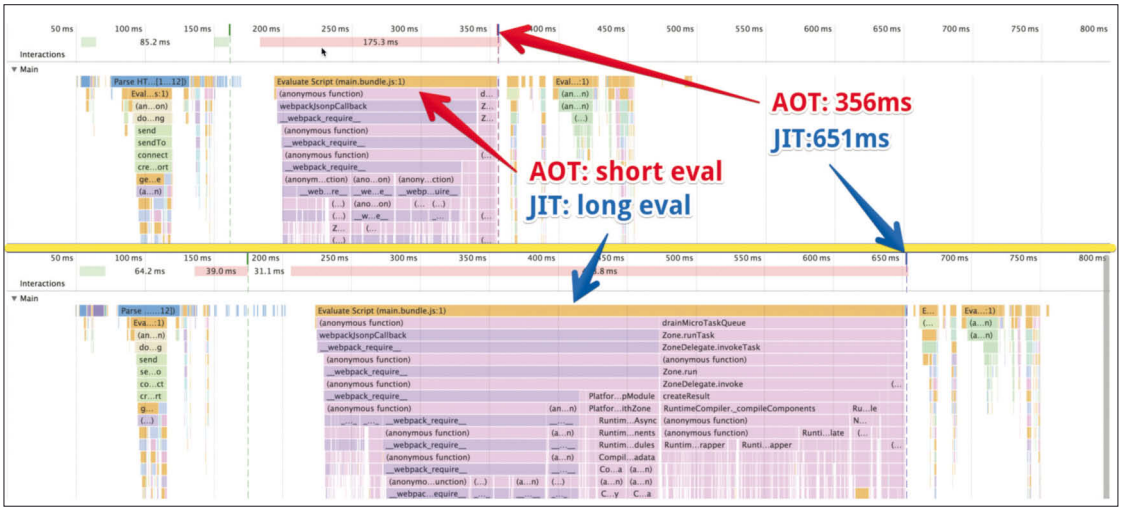


Figure 4

moyenne, il a été estimé qu'un TTI supérieur à 3 secondes risque d'obliger les utilisateurs à abandonner l'application Web. Vous pouvez mesurer cette valeur en utilisant les DevTools de votre navigateur préféré.

Pour les applications Angular, l'une des optimisations que nous pouvons appliquer pour réduire le TTI consiste à charger de manière différée les modules. De plus, ce chargement différé permet de réduire la taille initiale du *bundle* principale, ce qui à son tour contribue à réduire le temps de chargement.

```
export const routes: Route[] = [
 {
  path: '',
  redirectTo: '/home',
  pathMatch: 'full' },
 {
  path: 'home',
  loadChildren: () => import('../home/home.module').then(m => m.HomeModule)
 },
 {
  path: '**',
  redirectTo: '/home'
 }
];
```

Preloading (préchargement) des modules

Un des problèmes majeurs du chargement différé des modules Angular est qu'il pourrait ralentir la première navigation au sein de votre application, en fonction de la qualité du réseau, puisque le reste des modules est chargé à la demande. Si tel est le cas, vous pouvez toutefois précharger les modules concernés lors de la définition des routes :

```
import { RouterModule, PreloadAllModules } from '@angular/router';
RouterModule.forRoot([
 ...
], {
 preloadingStrategy: PreloadAllModules
});
```

Architecture de composants

Comment structurer les composants d'une application Angular de façon à ce qu'elle soit, réutilisable, faiblement couplée et performante ?

Une approche permettant de répondre à ces exigences techniques est de séparer les composants en composants conteneurs dits *smart* ou *container components* et composants de présentation aussi appelés *dumb* ou *prestantional components*. **Figure 5**

Les composants de présentation n'ont qu'une logique d'affichage, ils reçoivent des données en entrée, dont ils ne connaissant pas la provenance, et sont chargés de les afficher. Ils vont aussi éventuellement remonter des événements souvent déclenchés à partir d'interactions sur des éléments HTML sous-jacents (boutons, champ de texte, etc.).

Les composants conteneurs interagissent avec le reste de l'application à travers des services injectés et synchronisent les données entre leurs composants enfants. Prenons l'exemple d'une page CRUD classique de gestion d'utilisateurs. La meilleure façon d'imager cette décomposition en composants conteneurs et de présentation est de reproduire le visuel de la page en HTML, c'est ce qui est d'ailleurs préconisé de faire, car après tout, la finalité d'Angular est d'instancier les composants de façon déclarative depuis les templates HTML.

Le composant *user-management* (en violet) sera très souvent chargé à partir d'une règle de routage, il s'agira dans ce cas d'un composant conteneur et *Routed* en même temps. Bien que cette bijection soit régulière, elle n'est pas obligatoire. Le template du composant *user-management* ressemblerait à :

```
<alert [message]='message' [type]='type'></alert>
<user-form [user]='user' (save)='onSave($event)'></user-form>
<user-list [users]='users' (remove)='onRemove($event)' (edit)='user = $event'>
</user-list>
```

Le composant *user-management* a donc trois sous-composants :

- Le composant *alert* est un composant de présentation, il prend en entrée le message à afficher ainsi que son type (*error*, *warning* ou *success*). Le message sera déterminé par le *user-management* selon les événements sur la page.
- Le composant *user-form* est un composant de présentation également, il prend un modèle d'utilisateur en entrée représentant les données à mettre sur les champs de saisie du formulaire sous-jacent.
- Le composant *user-list* est un composant de présentation aussi, il récupère et affiche la liste des utilisateurs depuis le composant parent, il est aussi responsable de déclencher des événements liés aux actions faites sur une entrée de la liste des utilisateurs affichée.
- Le composant *user-management* est un composant dit

intelligent, il est chargé de décider de l'acheminement de la donnée entre ses composants enfants. Il injecte une instance du service de gestion des utilisateurs et selon l'action remontée par ses composants enfants, il exécute le bon appel API à partir du service injecté.

Comment penser réactif ?

Supposons que la liste des utilisateurs doit être automatiquement rafraîchie toutes les 30 secondes et qu'elle doit être paginée et filtrable à partir d'un champ de saisie.
Réfléchir de façon réactive consiste à réfléchir aux événements possibles sur l'interface sous forme de *streams* ou flux de données. À chaque événement correspond un flux.
Nous aurons ainsi :

- Un flux pour la saisie du filtre de recherche.
- Un flux pour la pagination.
- Un flux pour la *timer*.
- Un flux pour la récupération de la liste des utilisateurs depuis l'API.

L'implémentation réactive ressemblerait à :

```
page$: BehaviorSubject<number> = new BehaviorSubject(0);
ngOnInit() {
 const search$ = this.searchKeyword.valueChanges.pipe(
  startWith(''),
  debounceTime(500),
  tap(() => this.onPage(0))
 );
 this.list$ = combineLatest([search$, this.page$])
 .pipe(
  switchMap(([search, page]) => this.pollUsers(search, page))
 );
}
pollUsers(search: string, page: number) {
 return timer(0, this.STATUS_REFRESH_PERIOD_MS).pipe(
  switchMap(() => this.userService.getUsers(search, page))
 );
}
onPage(page: number): void {
 this.page$.next(page);
}
```

La syntaxe déclarative permet de facilement comprendre le flux des événements. Remarquez que nous n'avions eu besoin d'introduire aucun effet de bord dans notre code.
L'observable **this.list$** contient la liste d'utilisateurs à afficher selon les filtres et pagination appliqués. Nous appliquons ensuite le *pipe* **async** sur l'observable **this.list$** directement depuis le template qui en appelant la méthode *subscribe()* récupère chaque nouvelle donnée dans la liste des utilisateurs.
Quand le composant ainsi que le *pipe* sous-jacent sont détruits, le *pipe* appelle la méthode *unsubscribe()* sur l'observable **this.list$** évitant ainsi une fuite de mémoire notamment à cause du *timer* enregistré.

```
<user-list [users]='list$ | async'></user-list>
```

Attention aux appels de méthode depuis les templates

Quand nous attachons un appel de méthode dans le template, l'appel de cette méthode se fait à chaque cycle de détection de changement afin qu'Angular puisse comparer les anciennes et nouvelles valeurs, puis éventuellement mettre à jour la vue. Ceci peut dégrader considérablement les performances au *runtime* de l'application surtout si la méthode invoquée est gourmande en ressources.

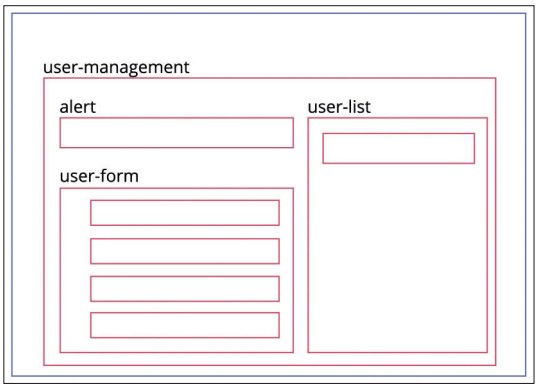


Figure 5

Très souvent, l'appel de méthodes peut être remplacé par une version réactive. Par exemple, dans un composant *user*, affichant les informations de l'utilisateur, au lieu d'écrire :

```
// user.component.html
getAge() {
 return Helpers.getAge(this.user.birthDate)
}

// user.component.html
<p> {{ getAge() }} </p>
```

Il est préférable de réagir au changement de l'utilisateur en *input* et préparer en amont la variable attachée. De cette façon, la méthode utilitaire **getAge()** est appelée uniquement au changement du modèle de l'utilisateur :

```
// user.component.html
age: number;
ngOnChanges(changes) {
 const { user } = changes;
 if (user) {
  this.age = Helpers.getAge(this.user.birthDate);
 }
}

// user.component.html
<p> {{age}} </p>
```

S'il s'agit d'un utilitaire de template commun à l'ensemble de l'application, il est préférable de l'écrire sous forme de *pipe* pure :

```
<p> {{ user.birthDate | age }} </p>
```

En savoir plus sur l'utilisation des *pipes(4)* en Angular.

Directives structurelles

Un besoin récurrent dans les applications de type **Admin** est la possibilité de contrôler l'affichage de certaines parties d'une page selon les rôles assignés à l'utilisateur connecté. Au lieu d'ajouter de la logique pour la gestion des accès au niveau du code TypeScript de chaque composant dont on souhaite contrôler l'affichage, il est préférable d'avoir une directive globale utilisable depuis n'importe quel template de l'application :

(4) https://angular.io/guide/pipes

```
<user-form *isAuthorized="'ADMIN'"></user-form>
<user-list *isAuthorized="'GUEST'"></user-list>
```

À l'image de ***ngIf** la directive ***isAuthorized** va décider de l'existence ou non d'un élément/composant dans le DOM selon une règle de gestion implémentée à l'intérieur de la directive. En effet, ***isAuthorized** est une directive structurelle, elle s'applique à un template et non à une vue concrète (composant ou élément DOM). Nous pouvons écrire notre directive de contrôle d'accès de la façon suivante :

```
import { Directive, Input, TemplateRef, ViewContainerRef } from '@angular/core';
@Directive({ selector: '[isAuthorized]' })
export class IsAuthorizedDirective {
  @Input() isAuthorized: string;
  constructor(private templateRef: TemplateRef<any>,
    private viewContainer: ViewContainerRef,
    private roleService: RoleService) {
  }
  ngOnInit() {
    if (this.roleService.isAuthorized(this.isAuthorized))
      this.viewContainer.createEmbeddedView(this.templateRef);
  }}
}
```

Headless components

Les *Headless components* ou plus communément connus sous le nom de *Render Prop* (dans l'écosystème React), est un pattern de composants utilisé dans plusieurs librairies graphiques. Le principe est de passer à un composant enfant possédant une certaine logique UI, un template personnalisé qui sera utilisé pour l'affichage des données résultant de cette logique. Prenons l'exemple d'un composant *ui-table* qui permet d'afficher les utilisateurs sous forme de tableau avec un sous-composant de pagination et un filtre de recherche :

```
<ui-table [loading]="true" [data]="users" (delete)="removeItem($event)">
 <ng-template let-user>
  <app-user [user]="user"></app-user>
 </ng-template>
</ui-table>
```

À partir de la donnée passée en paramètre, *ui-table* implémente la logique de l'affichage de la liste sous forme de tableau, fait apparaître un indicateur de chargement (*spinner*) lors du chargement et propose une option de pagination. Le composant *ui-table* donne la possibilité d'afficher chaque ligne du tableau comme bon nous semble, il suffit de lui passer le template HTML personnalisé. Ce composant va donc récupérer la donnée en paramètre et pour chaque ligne il va instancier le template personnalisé en lui passant l'objet courant de la liste parcourue (qui servira de context au template) :

```
// ui-table.component.ts
@Component({
 selector: 'ui-table',
 templateUrl: `ui-table.component.html`
})
export class UiTableComponent {
 @Input() data: any[] = [];
 @Input() loading = false;
 @ContentChild(TemplateRef) rowTemplate: TemplateRef<any>;
}

// ui-table.component.html
<div *ngIf="loading; else table">Some skeleton loader</div>
<ng-template #table>
 <div *ngFor="let row of data">
  <ng-container *ngTemplateOutlet="rowTemplate; context: { $implicit: row }">
  </ng-container>
  <ui-paginate [data]="data"></ui-paginate>
 </div>
</ng-template>
```

La propriété **rowTemplate** étant projetée, elle est récupérée via le décorateur **@ContentChild(TemplateRef).** La directive ***ngTemplateOutlet** permet d'instancier le template rowTemplate. L'option **{ $implicit: row }** permet quant à elle d'écrire **let-user** au lieu de devoir écrire **let-user="row"** avec un contexte **{ row: row }**.

Value Accessors

La fonctionnalité qui a sans doute fait la popularité d'AngularJS depuis sa première version est le fameux *two-way data binding*, l'effet était bluffant à l'époque ! En Angular, ceci peut être obtenu à partir du template via la propriété **ngModel** :

```
// user.component.ts
user = { name: '' };

// user.component.html
<input [(ngModel)]="user.name" /> {{ user.name }}
```

Ou encore sous sa forme réactive :

```
// user.component.ts
userControl = new FormControl();

// user.component.html
<input [formControl]="userControl" /> {{userControl.value}}
```

Il est possible d'utiliser la même notation et les mêmes fonctionnalités des formulaires Angular (*binding*, validation, état des inputs, *valueChanges*) sur des composants personnalisés, par exemple :

```
<form>
 <app-datepicker required [(ngModel)]="user.birthDate"></app-datepicker>
 <button [disabled]="form.invalid">Submit</button>
</form>
```

Ceci offre non seulement une simplicité d'écriture dans le template, mais associe également le *datepicker* au formulaire parent qui suit son état de validité : si la valeur du *datepicker* saisie/sélectionnée n'est pas valide, le formulaire sera invalide. Pour cela, il est important d'implémenter les méthodes de l'interface **ControlValueAccessor** proposée par Angular *Forms*. Voici un exemple complet :

```
import {Component, forwardRef, OnChanges, Input, OnInit} from
"@angular/core";
import {ControlValueAccessor, NG_VALUE_ACCESSOR} from "@angular
/forms";
@Component({
 selector: "app-date-picker",
```

```
 template: `<ui-date-picker [value]="value" (select)="onDateChange
($event)"> </ui-date-picker>`
 providers: [{
    provide: NG_VALUE_ACCESSOR,
    useExisting: forwardRef(() => DatePickerComponent),
    multi: true
 }]
})
export class DatePickerComponent implements ControlValueAccessor,
 OnInit, OnChanges {
 @Input() maxDate: Date = new Date();
 value: string;
 onDateChange(date: Date): void {
  this.onChange(date);
  this.onTouch(date);
 }
 writeValue(date: Date): void {
  if (date != null) {
   this.value = date;
   this.onDateChange(date);
  }
 }
 // Fonction de validation
 validate = (control: FormControl): null | { badDate: boolean } => {
  const value = control.value;
  return value < this.maxDate ? null : { badDate: true };
 };
 // Code obligatoire
 onChange: any = () => {};
 onTouch: any = () => {};
 registerOnChange(fn: any): void { this.onChange = fn; }
 registerOnTouched(fn: any): void { this.onTouch = fn; }
}
```

La méthode **writeValue()** sert à récupérer la valeur du modèle parent afin de l'utiliser dans la vue. Les appels à **onChange()** et **onTouch()** servent à remonter la valeur de la vue vers le composant parent qui l'affecte implicitement au modèle attaché, par exemple **(ngModelChange)="user.birthDate=event".**
En savoir plus sur l'interface *ControlValueAccessor*(5)**.**

Déploiement en production

Avec la sortie récente de la version 12 d'Angular, la commande **ng build** générera par défaut un build de prod (plus besoin d'ajouter **--prod**) ce qui permettra d'éviter de déployer des versions de dev en production. En ce qui concerne le déploiement en environnement de prod, le site officiel d'Angular propose un guide permettant de déployer nos apps Angular en production chez différents fournisseurs tels que AWS, Netlify ou encore Azure(6). Microsoft a d'ailleurs annoncé récemment un nouveau service, Azure Static Web Apps, permettant d'automatiser le déploiement ainsi que l'hébergement d'applications Web dites statiques.
Pour plus de détails, voici un guide complet pour déployer des apps Angular avec Azure Static Web Apps(7).

(5) https://angular.io/api/forms/ControlValueAccessor

(6) https://aka.ms/free-azure-fr

(7) https://aka.ms/angular-swa-fr

Rester à jour

À l'instar de nombreux projets, une version majeure d'Angular est prévue tous les 6 mois. Le but de ces versions majeures est l'ajout de nouvelles fonctionnalités ainsi que des mises à jour de sécurité et améliorations des performances internes au framework. Il est donc très important de rester à jour avec la dernière version d'Angular, ou à minima la version n-1. Angular CLI propose une mise à jour automatique d'Angular ainsi que ses dépendances (ceci ne concerne pas les dépendances tierces !) avec la commande **ng update.** Dans le meilleur des cas, la commande **ng update @angular/cli @angular/core rxjs** devrait vous aider à passer d'une version n à une version n+1. Dans certains cas, des erreurs liées aux *peerDependencies* peuvent être détectées par le CLI, souvent l'ajout de l'option **--force** permet de résoudre ces warnings. Il existe aussi un guide de mise à jour sur le site officiel(8).

Conclusion

Avec l'arrivée de la nouvelle architecture autour d'Ivy(9) et la façon dont le cœur d'Angular a été revu depuis la version 12, cela va permettre de tester de nouvelles techniques d'optimisation afin de réduire encore plus le temps de chargement des applications Angular, réduire le *Time-To-Interactive*, etc. Nous pouvons imaginer par exemple rendre les modules optionnels et ainsi charger les composants directement et individuellement (restera la dépendance vers Zone.js à résoudre !). Une autre piste à explorer est de pousser encore plus loin le *Server-Side Rendering*, en permettant d'avoir des applications qui seraient rendues côté serveur puis envoyées, avec leur *State* créé côté serveur, au client (navigateur Web, voire d'autres types de clients ?). Ainsi, du côté navigateur, Angular pourra simplement "reprendre" la main en se basant sur le DOM et le *State* générés par le serveur. Cela éviterait donc à Angular de "refaire" ce qui a déjà était fait par le serveur et de réhydrater le DOM (ce que fait l'implémentation actuelle du SSR, aka *Universal(10)*). Grâce à ces nouvelles techniques, il serait ainsi possible de booster radicalement le temps de chargement des applications Angular.
Le futur d'Angular sera donc marqué par un runtime de plus en plus léger, des outils de dev de plus en plus rapides, tels que le récent Angular DevTools(11), et des applications de plus en plus performantes.
La liste des préconisations étant assez longue, nous avons choisi de vous présenter un sous-ensemble dans ce dossier. Nous aurions pu parler de l'utilisation de Bazel pour les builds, la bonne gestion des *shared modules*, des *schematics* et *builders*, l'utilisation des *bundle analyzers*, de documentation, etc. Mais ce sera pour un prochain dossier. En attendant, nous serions ravis de répondre à toutes vos questions et retours et de vous aider à bien démarrer et évoluer avec Angular. N'hésitez pas à nous envoyer vos tweets sur nos comptes Twitter *@manekinekko (Wassim)* et *@chihabotmani* (Chihab).

(8) http://update.angular.io

(9) https://angular.io/guide/ivy

(10) https://medium.com/google-developer-experts/angular-universal-for-the-rest-of-us-922ca8bac84

(11) https://blog.angular.io/introducing-angular-devtools-2d59ff4cf62f

Y'a pas d'IDE plus simple que VSCode pour débuter en Java

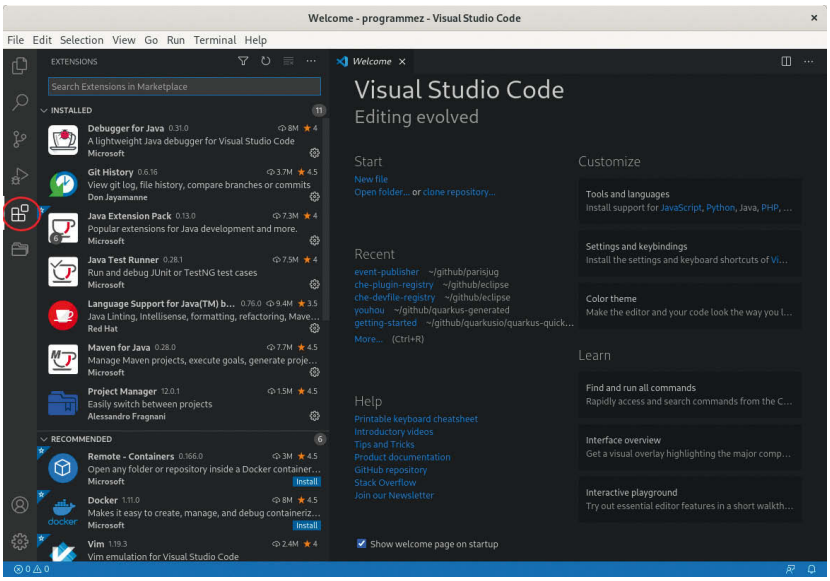
Sun Tan

Senior Software Engineer chez Red Hat, Sun contribue au projet Eclipse Che, un Cloud IDE open source. Ce développeur Java de plus de 15 ans d'expérience est également JUG Leader du Paris JUG dont l'objectif est de promouvoir Java à travers des présentations au format meetup à Paris.

Aujourd'hui, Eclipse IDE et IntelliJ IDEA sont les principaux environnements de développement utilisés pour les applications Java. Cependant, ces dernières années, un nouvel IDE a fait son apparition : Visual Studio Code (VSCode).

Un IDE est un logiciel dédié à la programmation. Il se présente généralement sous la forme d'un éditeur de code amélioré pour assister le développeur lors de ses principales tâches de programmation. Il permet au développeur d'être plus productif. Les IDE sont souvent spécialisés dans un langage de programmation. Initialement, VSCode était principalement destiné au développement des applications frontend. Depuis, la tendance a bien évolué et selon un récent sondage de Jaxenter (1), 27% des développeurs Java ont adopté VSCode comme IDE principal pour coder en Java.

Figure 1



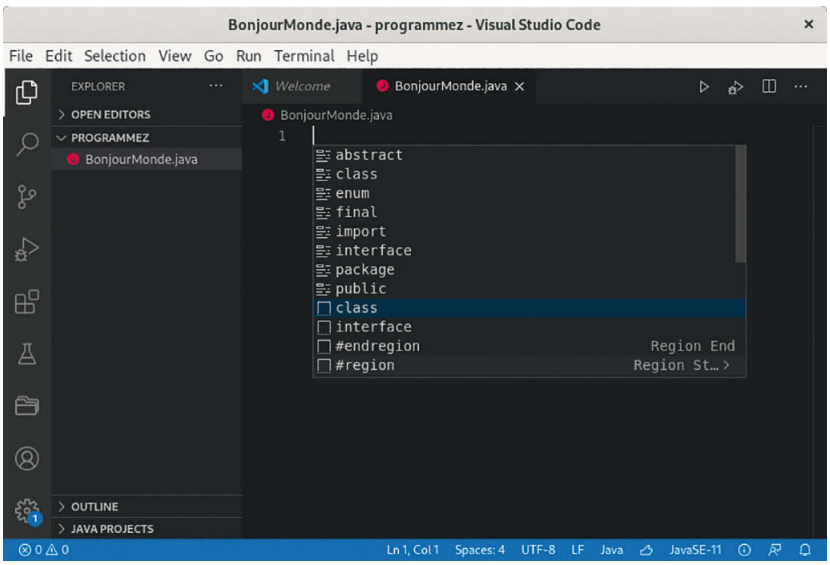


Figure 2

VSCode est un éditeur de code léger et se positionne à mi-chemin entre un éditeur de texte et un IDE. La première version est sortie en novembre 2015 à la suite des travaux d'Erich Gamma. Gamma est l'un des "Gang of Four", connus comme les auteurs de l'ouvrage "Design patterns. Catalogue des modèles de conception réutilisables" (2). Il est également à l'origine du moteur Java pour l'IDE Eclipse: Eclipse JDT.

VSCode est publié en open source par Microsoft et est disponible sous Linux, Windows et macOS en libre téléchargement (3).

VSCode propose un marketplace intégré pour l'installation d'extensions. Dans cet article, nous couvrirons principalement l'extension "Language Support for Java by Red Hat", qui offre les fonctionnalités d'autocomplétion, de validation et de coloration syntaxique en Java. Il existe également d'autres extensions Java pour les tests unitaires, le debug, etc. Avec presque 10 millions de téléchargements, "Language Support for Java by Red Hat" est l'extension Java la plus téléchargée sur le marketplace de VSCode. Il est possible d'installer l'ensemble des principales extensions pour Java via l'extension pack Java (4). L'installation des extensions et des packs d'extensions se fait à partir de la vue 'Extensions'. **Figure 1**

Pour plus de simplicité, dans la suite de cet article, nous utiliserons le nom "VSCode Java" pour désigner l'extension "Language Support for Java(TM) by Red Hat".

Débuter en Java ? Rien de plus facile

Pour débuter en Java, le développeur commence généralement par apprendre les notions et techniques de base en Java :

- gérer les variables ;
- définir des conditions ;
- créer des boucles ;
- créer des classes, méthodes et fonctions ;
- etc.

Les premières heures d'apprentissage du langage Java se traduisent souvent par la création de programmes composés d'une classe et d'une simple méthode statique `main` à compiler et exécuter.

Un `Hello world` en quelques secondes

VSCode simplifie le développement, l'exécution et le débogage du programme `main`. Il est possible de créer et exécuter son premier `Hello world` en quelques secondes. Une fois VSCode et Java installés sur votre poste (5):

1. Installez les extensions Java et Java Debug à partir de la vue `Extension`;
2. Ouvrez un nouveau dossier où vous souhaitez créer vos fichiers `.java`;
3. Créez un nouveau fichier `BonjourMonde.java` depuis la vue `Explorer`;
4. Le fichier créé est vide. Il faut attendre que le serveur Java soit démarré (pouce en l'air dans la barre du bas). Utilisez le raccourci `Ctrl-espace` et sélectionnez le snippet `class` pour générer le corps de la classe `BonjourMonde`. **Figure 2**
5. Dans le corps de la classe `BonjourMonde`, utilisez à nouveau le raccourci `Ctrl-espace` et sélectionnez le snippet `main`
6. Dans la méthode `main`, générez le code pour afficher un message sur la sortie standard avec le raccourci `Ctrl-espace` et le snippet `sysout` **Figure 3**
7. Une fois la méthode statique `main` générée, vous noterez des `Codelens` pour exécuter ou déboguer le programme. Cliquez sur `Run`. **Figure 4**
8. Bonjour Monde ! **Figure 5**

En quelques secondes, sans rien configurer, nous avons pu créer la classe `Main` et l'exécuter. Il en est de même pour le débogage.

Aucune configuration pour le débogage

Qui n'a jamais débogué du code Java en y insérant des instructions `System.out.println` ? Je dois avouer que cela m'arrive encore et régulièrement lorsque le débogage n'est pas configuré sur mon projet ou mon IDE. Avec VSCode, vos méthodes `main` seront également annotées par un Codelens `Debug`. Le lancement du mode debug est simplifié à l'extrême : il suffit d'ajouter un point d'arrêt sur la ligne, puis de cliquer sur le Codelens `Debug`. VSCode passe alors en mode Debug avec une vue dédiée au débogage. À partir de la vue `Debug`, le développeur a accès aux variables locales, à la pile d'exécution, etc. Il peut également modifier dynamiquement une des variables locales.

Aussi simplement que pour l'exécution d'un `main`, débugger ses premiers programmes Java est à la portée de tous avec VSCode. **Figure 6**

Ajouter une bibliothèque

Dans la plupart des projets, les dépendances de bibliothèques tierces en Java sont gérées traditionnellement par Maven ou Gradle. Mais ces outils nécessitent un apprentissage à part entière. Il vaut mieux ne pas effrayer les développeurs débutants avec de nouvelles notions comme le `dependencyManagement` ou le `pom` parent.

Avec VSCode, ajouter une bibliothèque à ses premiers programmes en Java est un jeu d'enfant.

Il suffit de :

- créer un dossier `lib`;
- d'y ajouter les fichiers `.jar` des dépendances tierces.

VSCode détecte alors les bibliothèques lors de l'édition de fichier Java et exécutera automatiquement le programme avec le bon classpath.

De plus, si le jar "sources" est disponible sur Maven Central,

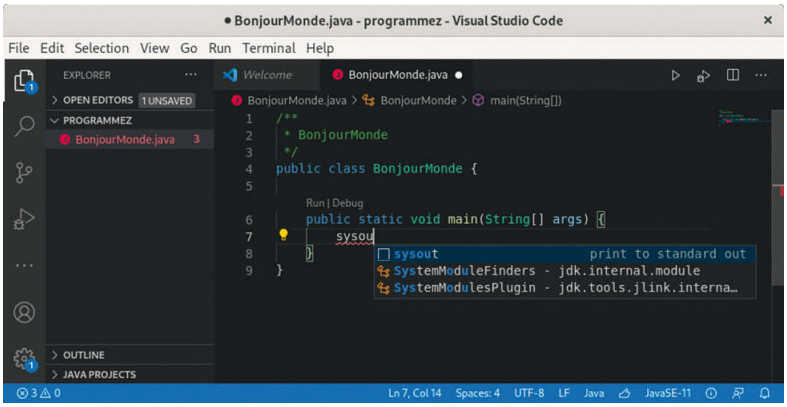


Figure 3

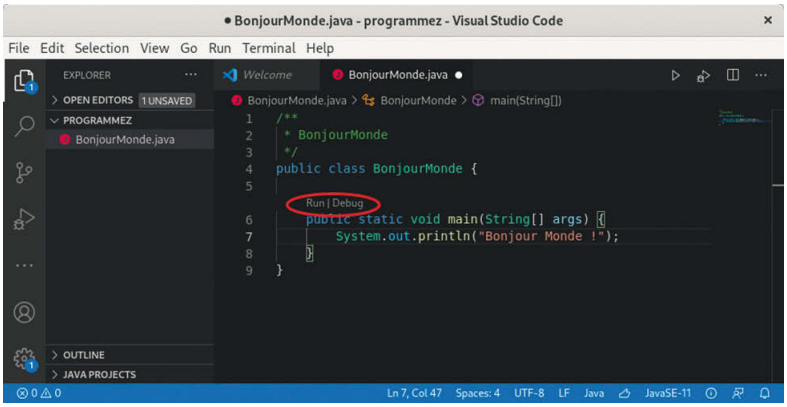


Figure 4

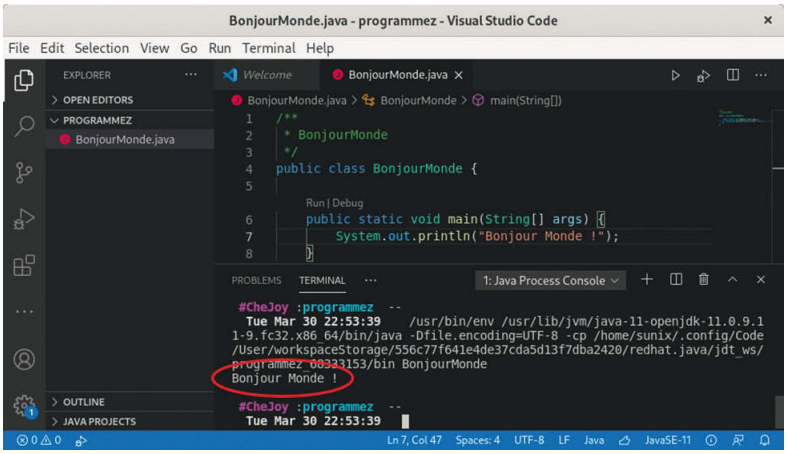


Figure 5

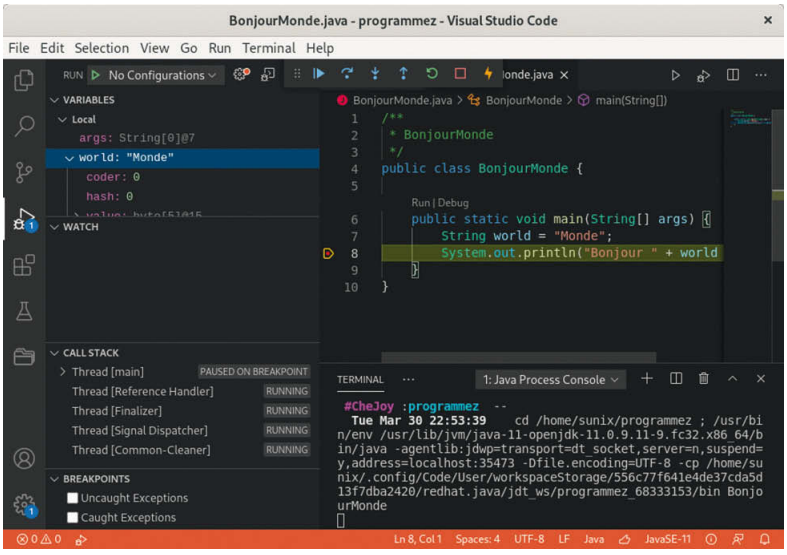


Figure 6

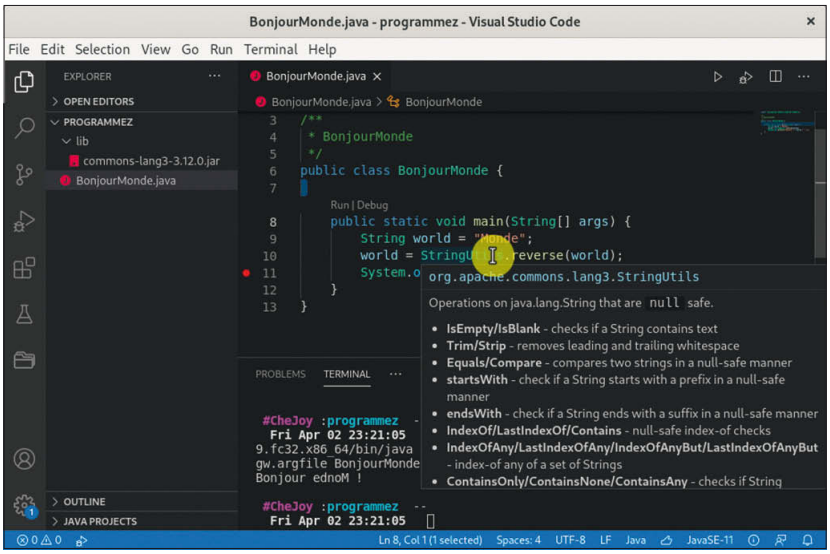


Figure 7

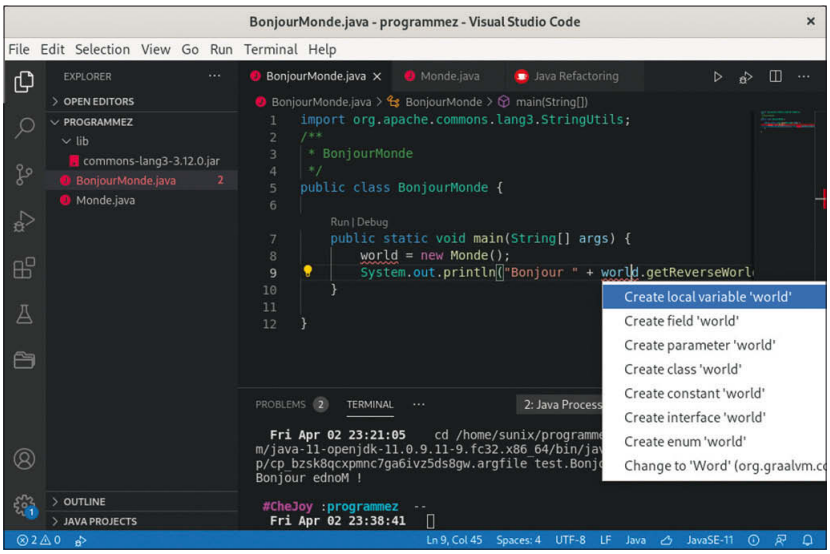


Figure 8

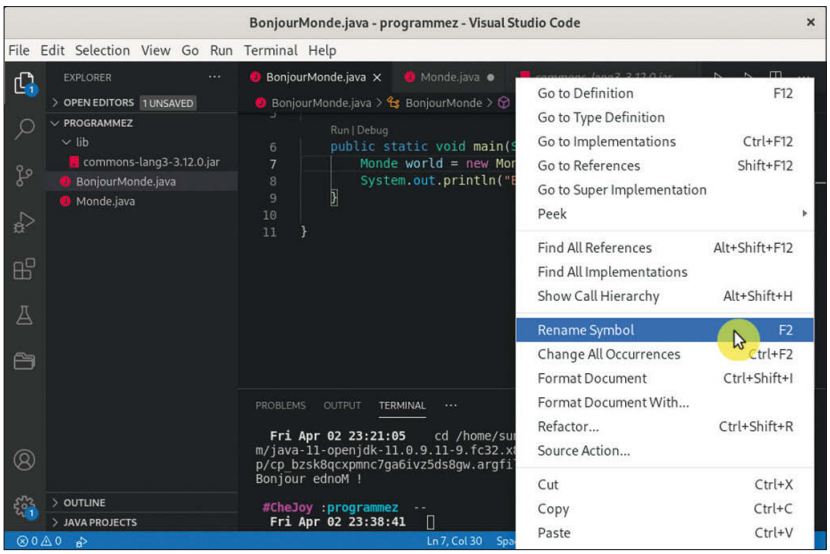


Figure 9

VSCode affiche la Javadoc lorsque l'on survole une classe venant de la bibliothèque. On peut également ajouter le fichier `-sources.jar` dans le dossier lib si les sources ne sont pas sur Maven Central. **Figure 7**

Léger et complet

VSCode Java n'a rien à envier aux autres IDE Java. Il est complet tout en restant léger. Comme vu précédemment, les fonctionnalités de complétion de code, snippets et Javadoc sont bien supportées.

VSCode Java vous permet également de trouver vos erreurs et de les corriger. Par exemple, lorsque vous utilisez une variable qui n'est pas définie, VSCode souligne la variable en rouge. En pressant `Ctrl-.` VSCode vous propose différentes solutions au problème comme la création de la variable locale. Dans les termes VSCode, il s'agit de `code actions` qui est l'équivalent du `QuickFix` dans Eclipse. **Figure 8**

VSCode Java supporte également le refactoring. Comme dans l'IDE Eclipse, il suffit d'un clic droit sur l'élément du code comme le nom d'une classe pour pouvoir accéder aux options de refactoring. On peut ensuite renommer ou déplacer une classe. Toutes les références à cette classe sont alors automatiquement rafraîchies. **Figures 9 et 10**

Sous le capot

VSCode Java n'a que quelques années d'existence. Pourtant il supporte déjà quasiment toutes les fonctionnalités que peut offrir un Eclipse IDE ou IntelliJ IDEA. Comment cela est-il possible ? La raison est que, sans le savoir, lorsque vous développez en Java sur VSCode, vous utilisez en fait Eclipse IDE. Décortiquons le fonctionnement des extensions dans VSCode et l'extension "Language Support for Java(TM) by Red Hat".

Language Server Protocol (LSP)

Une extension VSCode est développée en TypeScript. Pour y ajouter des fonctionnalités, on utilise l'API JavaScript de VSCode. Concernant les fonctionnalités liées à un langage, telles que la complétion de code ou l'affichage des erreurs, il existe un namespace `language` dans l'API de VSCode. Cependant, l'utilisation de TypeScript est une contrainte pour l'outillage souvent écrit dans son propre langage de programmation.

Pour remédier à ce problème, VSCode propose et intègre le Language Server Protocol (LSP). Le LSP est un protocole de communication basé sur JSON-RPC. Il définit les échanges entre un éditeur de code et un serveur de langage. Avec JSON-RPC, l'éditeur de code et le serveur peuvent être écrits dans des langages différents. Voici un exemple de communication entre un éditeur de code et un serveur de langage pour l'affichage d'erreurs dans un fichier. **Figure 11**

JDT.ls

Ainsi, VSCode Java implémente le LSP pour la partie Serveur de langage. Et pour ne pas réinventer la roue, l'implémentation choisie n'est rien d'autre que Eclipse JDT.

Eclipse JDT (Java Development Tools) est le moteur Java de l'IDE Eclipse utilisé pour toutes les fonctionnalités spécifiques au langage Java.

Ainsi, lorsque vous utilisez VSCode Java, une application Eclipse se lance en tâche de fond. Cette application Eclipse est composée d'Eclipse JDT, LSP4J et de quelques autres plug-ins comme Eclipse M2E. Évidemment, cette application Eclipse n'inclut pas les plug-ins non nécessaires au serveur de langage Java comme les plug-ins SWT en charge de l'interface graphique d'Eclipse.

Le projet, nommé JDT.ls pour JDT language server, est disponible sur GitHub (6) en open source. Pour ajouter le support de Java à un éditeur de code supportant le LSP, il suffit de packager correctement JDT.ls.

Autres IDE utilisant JDT.ls

Finalement, on peut dire que VSCode n'est qu'une nouvelle interface utilisateur d'Eclipse JDT en plus de l'IDE Eclipse. Et pourquoi ne pas développer une interface Web pour Eclipse JDT ?

Pour ne rien vous cacher, je suis un des contributeurs du projet Eclipse Che (7) : un IDE pour lancer des workspaces dans le cloud (Kubernetes ou Openshift) et coder sur son navigateur. Che intègre l'extension VSCode Java dans ses plug-ins Java et Quarkus.

Par ailleurs, Che est bien plus qu'un browser IDE : il permet d'appliquer les principes du IaC (Infrastructure as Code) pour les environnements de développement. On parle alors de Developer Environment as Code (DEaC). Cela consiste à pouvoir lancer des environnements de développement à la demande et de les répliquer juste à partir d'un fichier de description : le devfile.yaml (8). Ainsi, le développeur peut rapidement générer des environnements de développement à la demande sur ses différents projets et éviter les effets du type "Mais... ça marche sur mon poste".

D'autres IDE "Cloud" utilisent également VSCode Java :

- Eclipse Theia (9) : éditeur par défaut d'Eclipse Che et Gitpod.io
- CodeReady Workspaces : la version Red Hat d'Eclipse Che que vous pouvez tester sur la Red Hat Developer Sandbox (10)
- Gitpod.io (11)
- Github Codespaces (12)

Enfin le LSP permet de bénéficier du support Java sur des éditeurs de code comme Emacs (13) ou Vim (14). Vous trouverez plus d'informations sur le site du LSP (15) publié par Microsoft. On y retrouve notamment la liste des outils supportant le LSP (16).

Conclusion

L'un des points forts de VSCode est son écosystème qui n'est pas restreint à un unique langage. La communauté open source a su bâtir des extensions de qualité pour les langages et frameworks que vous utilisez au quotidien. J'espère vous avoir convaincu de la simplicité de VSCode pour développer vos premières applications Java. Bien sûr, VSCode supporte tout aussi bien les projets Java à l'échelle de l'entreprise.

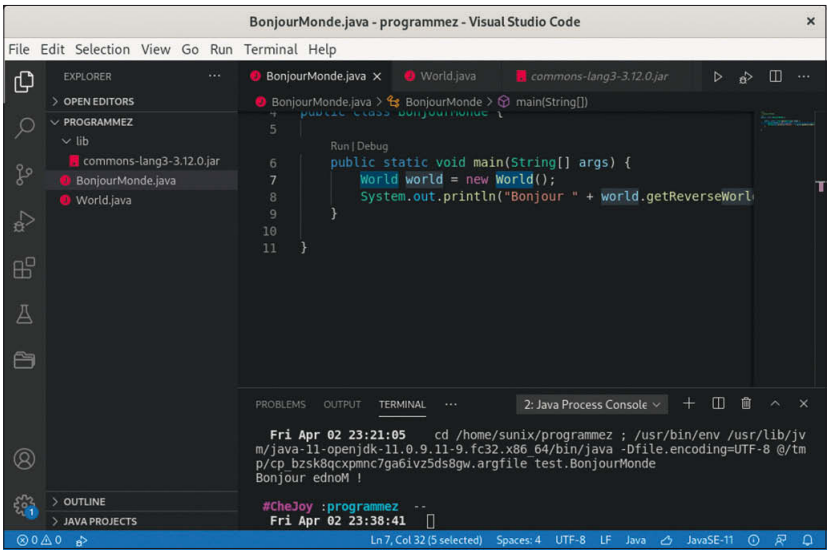


Figure 10

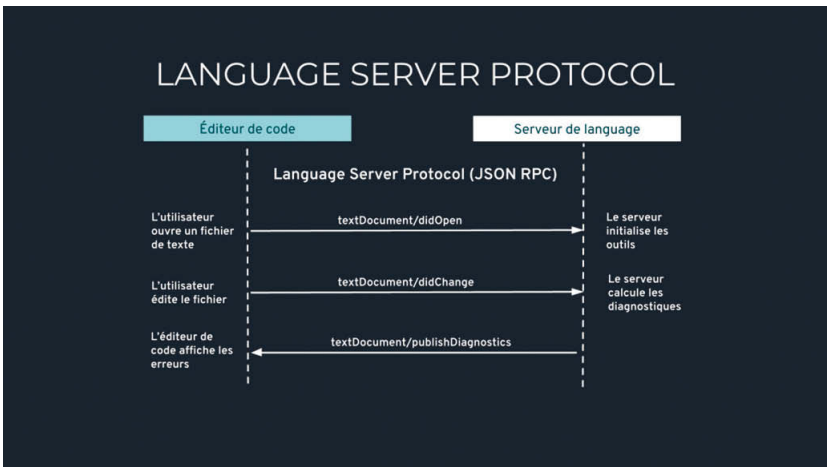


Figure 11

Les liens :

(1) https://jaxenter.com/java-development-2021-173870.html
(2) https://code.visualstudio.com/docs/languages/java
(3) https://code.visualstudio.com/download
(4) Extension pack Java : https://code.visualstudio.com/docs/languages/java
(5) https://code.visualstudio.com/docs/java/java-project#_configure-jdk
(6) https://github.com/eclipse/eclipse.jdt.ls
(7) http://eclipse.org/che
(8) Spécification du devfile: https://devfile.github.io/
(9) https://theia-ide.org/
(10) https://developers.redhat.com/developer-sandbox
(11) https://www.gitpod.io/
(12) https://github.com/features/codespaces
(13) https://github.com/emacs-lsp/lsp-java
(14) https://github.com/lgranie/vim-lsp-java
(15) https://microsoft.github.io/language-server-protocol/
(16) https://microsoft.github.io/language-server-protocol/implementors/tools/

Yohan Lasorsa
Senior Cloud Advocate
@ Microsoft
@sinedied

Un environnement de dev propre et toujours à jour grâce aux conteneurs et VS Code

Parmi les choses qui me font perdre le plus de temps en tant que dev et qui deviennent facilement source de frustration, en voici trois :

- Installer sa machine
- Mettre à jour son environnement de dev
- Réinstaller sa machine, parce qu'à force d'installer et mettre à jour ses outils de dev, plus rien ne marche :-(

La configuration présentée dans cet article fonctionne sur les OS supportés par VS Code : Linux, macOS, Windows. Au préalable, il faut que VS Code et Docker soient installés sur la machine. Nous n'avons pas testé sur ARM.

Et si je vous disais qu'on peut s'éviter tous ces problèmes, et avoir un environnement dev isolé pour chacun de vos projets, partageable au sein d'une équipe, et qui se met à jour facilement, ça vous tente ?

C'est maintenant possible avec l'extension **Remote Development Extension** pour VS Code, téléchargeable sur https://aka.ms/vscode/remote-dev, qui va vous permettre de configurer votre environnement de dev dans un conteneur. Dans cet article, nous verrons en détail sa mise en place sur un projet et les avantages associés.

Pourquoi développer dans un container

Dans chaque projet logiciel, l'histoire d'un nouveau développeur rejoignant l'équipe est (presque) toujours la même :

- « Salut, bienvenue dans l'équipe ! On est ravi de t'avoir avec nous ! »
- (Un peu plus tard, après les présentations de rigueur) « Voilà la doc pour installer ton environnement pour le projet. Quand tu seras prêt à attaquer, fais-nous signe ! »

Bon parfois, *la doc n'existe pas*, et ce seront vos collègues qui vous diront quoi installer. Très probablement, quelques heures (ou jours) plus tard, la suite de l'histoire est la suivante :

- « À l'aide, je ne comprends toujours pas pourquoi le projet ne veut pas compiler et se lancer ! »

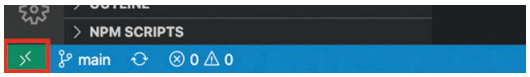
J'ai été ce développeur, plus d'une fois déjà. Après avoir revu tout ce qui a été installé, on découvre en général l'un ou plusieurs des problèmes suivants :

- La doc d'installation n'est pas à jour (étonnant, non ?). Ce n'est pas justement le boulot du nouvel arrivant de la mettre à jour ?
- Vous avez installé les bons outils, mais pas *exactement* la bonne version. « Désolé, on a oublié de te dire qu'il faut absolument la version 3.1.16.beta 2 de la base de données, on a quelques soucis dans le code. »
- Votre environnement existant configuré pour d'autres projets rentre en conflit avec votre nouvelle installation.

Ces problèmes, on les connaît bien puisque ce sont *exactement* les mêmes que l'on peut avoir lors de la mise en production de nos applications. La bonne nouvelle, c'est qu'on sait très bien les résoudre en packageant l'environnement d'exécution dans des conteneurs, à l'aide par exemple de Docker. Dans ce cas, on pourrait mettre en place la même chose pour notre environnement de dev, non ? C'est exactement ce qu'on va faire !

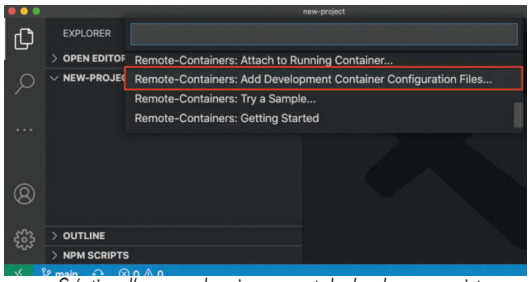
Mise en place sur un projet

Vous aurez besoin de ces outils installés sur votre machine pour pouvoir démarrer : **Visual Studio Code**, **Docker** et l'extension **Remote Development**. Ouvrez un projet existant dans VS Code (ou créez en un nouveau), et cliquez ensuite sur l'icône en bas à gauche :



Activation de l'extension dans VS Code

Ensuite, sélectionnez **Add Development Container Configuration Files**, et vous allez avoir une liste d'environnements de départ qui va s'afficher.



Création d'un nouvel environnement de dev dans un projet

Il existe des configurations préconstruites pour Node.js, Java, .Net, Go, Python et autres, vous en trouverez une à coup sûr qui pourra servir de point de départ pour votre projet. Après avoir fait votre choix, vous verrez apparaître un nouveau dossier **.devcontainer**, qui contient deux fichiers :

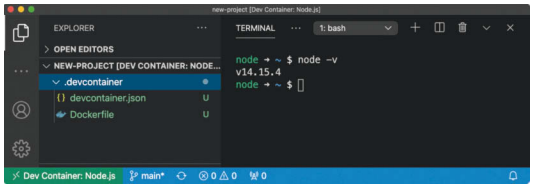
Dossier contenant la configuration de l'environnement de dev

On va regarder en détail ces fichiers juste après, mais d'abord on va recharger notre projet pour travailler cette fois dans notre conteneur de dev.

Cliquez une nouvelle fois sur l'icône en bas à gauche dans VS Code, et sélectionnez cette fois **Reopen in container**.

VS Code va recharger la fenêtre de votre projet et construire le conteneur, ce qui peut prendre quelques minutes au premier lancement. Par la suite, le chargement sera quasi-ins-

tantané. Une fois terminé, vous pourrez constater que VS Code est connecté à votre conteneur de dev via la barre de statut.



VS Code connecté à un container de dev Node.js

Vous pouvez maintenant ouvrir un terminal et utiliser votre nouvel environnement de dev ! Dans mon exemple, j'ai configuré mon environnement pour utiliser Node.js en version 14. Notez que si maintenant tous mes outils de dev sont bien isolés dans mon conteneur, le code source du projet ne s'y trouve pas : il reste stocké directement sur ma machine, et connecté au container via un *volume*. Il n'y a donc aucun risque de perdre des données si le conteneur est détruit.

Personnalisation de l'environnement de dev

Maintenant que vous avez un environnement de travail de base, vous voudrez certainement le compléter pour les besoins spécifiques de votre projet. Si vous dépliez le dossier **.devcontainer** dans l'explorateur, vous verrez ces deux fichiers qui vous permettront de le faire :

- **Dockerfile** : ce fichier spécifie la configuration de votre conteneur et permet de construire l'image qui sera utilisée pour votre environnement. C'est un fichier d'image Docker classique. Si vous devez installer des outils supplémentaires ou configurer des variables de scripts et d'environnement pour votre environnement de développement, c'est ici qu'il faut le faire.
- **devcontainer.json** : ce fichier permet de personnaliser VS Code lorsque le projet est connecté au conteneur de dev. En particulier, il vous permet de changer les paramètres de VS Code, ajouter des extensions à installer automatiquement, configurer des redirections de port ou encore exécuter des commandes au démarrage du conteneur.

Regardons maintenant plus en détail comment personnaliser votre environnement de dev sous VS Code en modifiant le fichier **devcontainer.json**.

Sous la clé **settings**, vous devriez voir quelque chose comme cela :

```
// Set *default* container specific settings.json values on container create.
"settings": {
  "terminal.integrated.shell.linux": "/bin/bash"
},
```

Cette clé vous permet de surcharger n'importe quel paramètre de VS Code lorsque celui-ci est connecté au conteneur. Par exemple, si vous travaillez sous Windows et que votre terminal par défaut est PowerShell, en vous connectant au conteneur de dev le paramètre ci-dessus va changer votre terminal par défaut par Bash, qui sera exécuté dans le système Linux du conteneur. Grâce à ces paramètres, vous pouvez définir un formateur de code commun à l'équipe ou forcer la signature des commits par exemple.

Une autre option très utile se trouve sous la clé **extensions** :

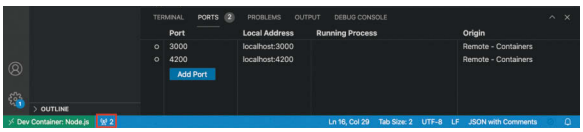
```
// Add the IDs of extensions you want installed when the container is created.
"extensions": [
  "dbaeumer.vscode-eslint"
],
```

Celle-ci va vous permettre de définir une liste d'extensions qui seront installées automatiquement *dans* le conteneur lors de son premier lancement. Oui, vous avez bien lu : ces extensions ne viendront pas polluer votre configuration VS Code globale, car elles seront disponibles *uniquement* lorsque le conteneur sera lancé et connecté à votre projet. Plus besoin de jongler avec les extensions selon le projet, fini l'extension C# qui ne sert à rien sur votre projet Java !

Une dernière option indispensable, **forwardPorts**, va vous permettre de rendre des ports disponibles à l'intérieur de votre conteneur, accessibles par défaut depuis votre machine locale.

```
// Use 'forwardPorts' to make a list of ports inside the container available
locally.
"forwardPorts": [3000, 4200],
```

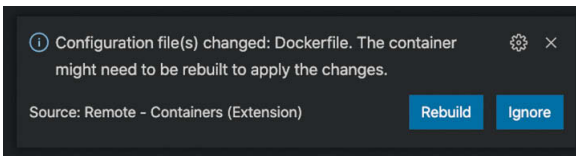
Par exemple, si vous lancez un serveur dans un conteneur, sans transfert de ports il ne sera pas possible d'y accéder depuis votre navigateur. À noter qu'il est possible de rajouter des ports à transférer à la volée en cliquant sur l'icône d'antenne dans la barre de statut, puis en sélectionnant **Add Port** :



Le panneau de transfert de port dans VS Code

Partage et mise à jour de l'environnement

Une fois la configuration du conteneur et de VS Code terminée, il ne vous reste plus qu'à pousser ces deux fichiers dans votre dépôt de code, pour que toute votre équipe puisse en profiter. Et c'est bien là que se trouve l'un des gros avantages de cette approche : pour accueillir un nouveau développeur, il lui suffit désormais de cloner le projet et de recharger son VS Code pour utiliser le conteneur. Le temps d'un café (ou plusieurs selon la vitesse de connexion), il aura un environnement de dev tout prêt pour travailler, sans effort. Encore mieux, si vous mettez à jour la configuration en changeant la version d'un outil par exemple, vos collègues verront cette notification :



Il suffira de cliquer sur **Rebuild** pour mettre leur environnement à jour. Finis les retours de vacances difficiles ou plus rien ne marche !

Pour aller plus loin

Cet article n'offre qu'un bref aperçu de ce qu'il est possible de faire avec ces outils, et on peut bien évidemment pousser un peu plus au besoin, comme configurer un environnement de dev à base de plusieurs conteneurs.

Je vous invite à regarder la documentation à cette adresse pour les scénarios plus avancés : aka.ms/remote-dev-ext/advanced.

Et si vous voulez faire un bond dans le futur, jetez un œil à GitHub Codespaces, qui vous proposera bientôt tout cela… directement depuis votre navigateur, sans rien avoir à installer sur votre machine !

On refait le patch : l'outillage Lombok

L'outillage Lombok est présent dans de nombreux projets Java, en effet plus d'un million de personnes se connectent sur le site de Lombok chaque année. Pourtant, le choix de l'utiliser ou non suscite de nombreux débats entre développeurs, mais qui la plupart du temps ne dépassent pas le cadre du subjectif ("j'aime" ou "j'aime pas"). Nous avons consulté des développeurs Java chevronnés pour construire un argumentaire objectif des atouts et faiblesses de la bibliothèque fondée sur nos expériences, anecdotes et traumatismes divers.

Arnaud Thiefaine
Software Crafter
Arolla
Twitter:@ArnaudThiefaine
Github: athiefaine

Jimmy Tran
Software Crafter
Arolla
Github: jtran1403

Qu'est-ce que Lombok ?

Lombok est une bibliothèque open source Java qui a fait ses débuts en 2009. Dix ans après sa publication, Roel Spilker et Reinier Zwitserloot rappellent l'origine de Lombok en quelques mots :

« If there's too much code. And it don't look good. Who you gonna call? Boilerplate Busters! »[1]

Le but de la bibliothèque est d'éliminer le plus de code qualifié de boilerplate, c'est-à-dire du code verbeux, mais offrant peu de valeur ajoutée et ayant tendance à générer beaucoup de bruit dans le code des classes Java. Les éléments suivants sont particulièrement visés : accesseurs (*getters* et *setters*), constructeurs, égalité entre deux objets (méthode *equals()*), calcul du hashCode et rendu textuel (méthode *toString()*).

L'utilisation habile d'annotations permet de se passer de ces lignes de code superflues. Le procédé n'est toutefois pas magique, parce que bien que ces lignes ne soient plus visibles, elles continuent à exister. L'astuce consiste à les rendre absentes dans les sources, mais présentes dans le code compilé. Ainsi, avant la phase de compilation du code source, une phase de build est ajoutée par Lombok, qui parcourt les annotations dédiées pour générer le bytecode correspondant.

(1) Les auteurs de Lombok se comparent, en cherchant à éliminer le code boilerplate, jugé inutile, aux célèbres chasseurs de fantômes des films Ghostbusters. Voir début de la vidéo : https://devoxx.be/schedule/speaker-details/?id=25051

Un gain de productivité et de lisibilité

Lombok s'occupe d'écrire pour nous une bonne partie du code rébarbatif. Prenons pour exemple de la classe Person, traditionnellement implémentée de la manière suivante :

```
public class Personne {
  private String fullname;
  private String birthday;
  private List<Personne> contacts;

  public Personne() {
  }

  public String getFullname() {
    return fullname;
  }

  public void setFullname(String fullname) {
    this.fullname = fullname;
  }

  public String getBirthday() {
    return birthday;
  }

  public void setBirthday(String birthday) {
    this.birthday = birthday;
  }

  public List<Personne> getContacts() {
    return contacts;
  }

  public void setContacts(List<Personne> contacts) {
    this.contacts = contacts;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Personne personne = (Personne) o;
    return Objects.equals(fullname, personne.fullname) &&
```

```
        Objects.equals(birthday, personne.birthday) &&
        Objects.equals(contacts, personne.contacts);
  }

  @Override
  public int hashCode() {
    return Objects.hash(fullname, birthday, contacts);
  }

  @Override
  public String toString() {
    return "Personne{" +
        "fullname='" + fullname + '\'' +
        ", birthday='" + birthday + '\'' +
        ", contacts=" + contacts +
        '}';
  }
}
```

Ce code peut, avec Lombok, se simplifier ainsi :

```
@Data
public class Personne {
  private String fullname;
  private String birthday;
  private List<Personne> contacts;
}
```

Le gain en concision est indubitable, et surtout, le code n'a pas perdu en clarté. Bien au contraire, l'information recherchée est plus facilement accessible, et cela nous permet de nous concentrer sur l'essentiel. Côté écriture du code, l'effort à fournir se limite à utiliser les bonnes annotations.

Un gain énorme à la maintenance

L'argument suivant nous a été opposé : le gain en productivité était surtout vrai il y a 10 ans, mais actuellement les IDE (et notamment IntelliJ) sont capables de générer automatiquement tout ce code, et même de le masquer si besoin.

Cet argument est valable, notamment lors de l'écriture initiale de la classe. Par la suite, lorsqu'on va vouloir ajouter des attributs, ou refactorer en déplaçant des attributs vers d'autres classes ou en découpant la classe, les choses vont se compliquer, notamment pour conserver la cohérence des méthodes *equals()*, *hashCode()*, et *toString()*. La raison est que ces méthodes, contrairement aux accesseurs, mettent en œuvre l'ensemble (ou au moins plusieurs) des attributs de la classe. Chaque modification de l'ensemble des attributs d'une classe entraîne ainsi un effort de maintenance pour ces méthodes.

Il faut en effet les régénérer, mais est-ce si compliqué ? L'IDE sait le gérer, mais seulement à la demande. Le risque réside donc surtout dans la possibilité d'oublier de les régénérer (et donc par cet oubli d'introduire des bugs).

Lombok, en régénérant le code *boilerplate* à chaque compilation, fournit une aide importante pour éliminer cette charge mentale. Là où Lombok est très fort, c'est que peu importe l'évolution des attributs d'une classe, le code *boilerplate* sera toujours cohérent.

De manière analogue, d'autres annotations permettent d'avoir une classe immutable facilement, ou encore de gérer les builders/constructeurs automatiquement.

Montée en compétence sur Lombok

Lombok rend plein de services, mais en le faisant sous le manteau, au risque de paraître magique pour certains. Pire encore, d'autres pourraient totalement ignorer ce que fait Lombok (par exemple un membre de l'équipe débutant en Java dont le premier projet utilise Lombok).

Pour que l'utilisation de Lombok soit efficace et n'entraîne pas d'effet pervers, il est important que les coéquipiers soient alignés et au même niveau de connaissance, à la fois sur l'outil Lombok que sur les bonnes pratiques d'utilisation sur le projet.

Cela passe par une bonne connaissance du sens des annotations et du code implicite qu'elles vont produire. Par exemple, il est important de savoir que l'annotation @Value va appliquer les modificateurs *private* et *final* sur les attributs de la classe, de façon à garantir au mieux leur immutabilité.

Il reste important de préciser que s'approprier Lombok ne nécessite pas d'effort surhumain : effectivement, Lombok ne regroupe qu'une dizaine d'annotations et on en utilise en général qu'une poignée. Mais cet effort, bien que minime, reste nécessaire pour éviter les mauvaises surprises.

Impacts de Lombok sur le design

Comme mentionné ci-dessus, l'annotation @Value est idéale pour aider à rendre une classe immutable et aide donc à l'écriture des Value Objects, définis par la valeur de leurs attributs et non plus par la notion d'instance d'objet. L'annotation @Data est parfaite quant à elle pour les classes de type Data Transfer Objects (classes sans comportement servant à véhiculer de la donnée structurée).

Il est important de bien choisir les annotations Lombok en fonction de la situation. Par exemple, @Data est parfois utilisée comme l'annotation à tout faire (elle équivaut à la somme de @Getter, @Setter, @ToString, @EqualsAnd HashCode and @RequiredArgs Constructor). Mais au final avons-nous besoin de tout ça ? La question est importante, parce qu'une fois le choix effectué, il est plus difficile de revenir en arrière ("s'ils ont mis @Data, ce doit être pour une bonne raison, non ?"). L'usage des annotations doit se faire avec parcimonie, pour répondre à un besoin, et pas l'inverse (ne pas essayer de couvrir par défaut tous les besoins qui pourraient un jour se présenter)[2].

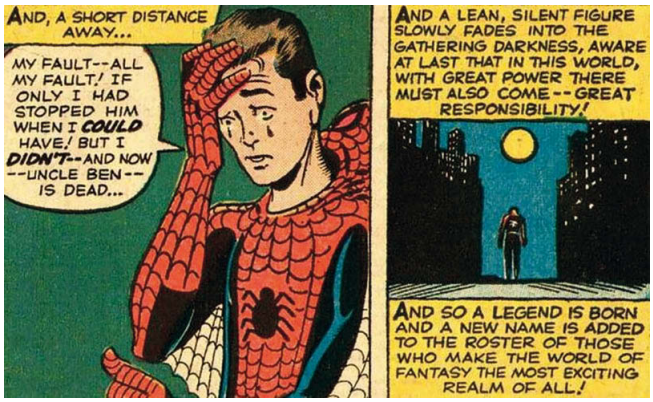
Par ailleurs, un des critères qui nous permet de savoir si une classe est bien découpée est son nombre d'attributs. Cela était particulièrement visible avec le nombre de lignes que la classe occupait, mais ce n'est plus forcément le cas avec l'utilisation de Lombok. Même si la classe ne fait plus que quelques lignes, il est important de ne pas sous-estimer sa complexité réelle.

Impacts de Lombok sur l'écosystème du projet

Lombok est en apparence simple à installer, il suffit d'ajouter la dépendance au projet. Les projets Java utilisent d'autres bibliothèques qui produisent du code invisible tel que Jax-RS pour les endpoints de l'API pour éviter de la maintenance ou bien MapStruct. Le risque de conflit lorsque ces dépendances génèrent le bytecode et interagissent entre elles est difficile à prévoir, surtout lors de leurs montées de version.

Par ailleurs, l'intégration de Lombok dans le projet crée un

(2) D'une certaine manière, on rejoint ici le principe YAGNI (You Ain't Gonna,Need It) cher aux artisans développeurs.



couplage supplémentaire. Comme toute brique tierce, la question du support à long terme reste légitime. Par le passé, il est même arrivé que Lombok freine la montée de version de Java : ainsi, à la sortie de Java 9, Lombok ne supportait pas encore totalement le nouveau JDK, et la compilation des projets avec Lombok échouait (voir changelog de Lombok). De la même façon, certaines incompatibilités avec l'IDE étaient parfois à noter, de même que la nécessité d'installer un plugin spécifique lui permettant d'interpréter les annotations Lombok.

Les angles morts du code implicite

Une fois familiarisé avec Lombok, on en vient rapidement à oublier sa présence. Lombok sait tellement se faire tout petit qu'en cas de problème, on aura du mal à penser qu'il peut être fautif. Dans les anecdotes des développeurs interrogés, il n'était pas rare d'avoir perdu plusieurs jours pour comprendre la cause d'un problème, en particulier parce que l'utilisation de Lombok en était l'origine.

Lombok, c'est du code qui fait partie de l'application, mais qui n'est plus visible directement. Les impacts sont les suivants :

- lors du debug, le code généré n'est pas directement sous nos yeux ; de plus il sera difficile voire impossible, de poser un point d'arrêt à l'intérieur du code généré par Lombok
- des limitations sont à prévoir dans la navigation du code, la documentation et la recherche d'une méthode

Malgré cela, les bugs liés à Lombok sont très rarement imputables à la bibliothèque elle-même, mais proviennent plutôt de mauvaises utilisations ou d'une mauvaise compréhension de son fonctionnement. Quelle que soit la cause réelle, de tels dysfonctionnements restent difficiles à qualifier.

Ironie du sort, les auteurs de Lombok ont eux-mêmes été confrontés à ce genre de difficulté. Ainsi, dans leur vidéo de feedback à l'occasion des 10 ans de l'outil, ils partagent leur difficulté à maintenir le projet open source en prenant pour exemple la correction d'un bug remonté par un utilisateur. L'origine du problème était une *race condition* dans leur code liée à une variable globale. La correction leur a pris 6 mois, en particulier à cause de la difficulté de reproduction du bug.

Grands pouvoirs = grandes responsabilités

La part de l'implicite dans un projet logiciel n'est pas l'exclusivité de Lombok. Certains langages (ou versions de langages), frameworks ou autres outils, produisent beaucoup d'implicite à travers leur déclarativité. Dans l'écosystème Java, les exemples les plus connus sont Spring ou Hibernate. Les services offerts sont immenses, offrant ainsi aux développeurs un immense pouvoir, mais leur fonctionnement paraît magique voire obscur pour certains. Certains font l'effort de monter en compétence pour maîtriser leur plate-forme, d'autres restent dans une situation où ils ne comprennent pas ce qu'ils font. Avec des conséquences dramatiques en production, telles que des failles de sécurité ou des performances désastreuses.

Ce n'est pas parce que ces outils font une grande partie du travail à la place du développeur que ce dernier doit oublier les fondamentaux et ignorer ce qu'il se passe sous le capot. Ainsi, les choix et l'utilisation doivent être faits en connaissance. Comme le disait Ben Parker[3], "un grand pouvoir doit s'accompagner de grandes responsabilités".

En l'occurrence, la responsabilité du développeur est de fournir les efforts nécessaires pour connaître et maîtriser ses outils.

Et ailleurs, c'est comment ?

En Kotlin, les getters/setters sont implicites en fonction du niveau de visibilité du champ et il est possible de qualifier sa classe de *data class* ou de *case class*.

En langage Go, l'accessibilité de l'attribut se définit par sa casse (par exemple, une majuscule pour rendre l'attribut publique).

En Groovy, les accesseurs existent par défaut sans devoir les définir. D'ailleurs Groovy supporte implicitement la plupart des fonctionnalités que Lombok offre à Java.

Ceux qui développent dans ces langages ne semblent pas mécontents de ces facilités, pourtant similaires à ce qu'apporte Lombok. Ces comportements sont natifs au langage et font donc partie des connaissances communes à tous les développeurs. En outre, ces fonctionnalités sont présentes gratuitement, sans ajouter de couplage supplémentaire.

Java semble d'ailleurs suivre cette direction en introduisant, en version 14, les *records*. Ceux-ci permettent de créer des classes immutables à partir d'une définition simplifiée, rendant inutile l'annotation @Value. En faisant preuve d'un peu d'optimisme, on peut imaginer que les prochaines versions de Java rendront Lombok totalement caduque...

Finalement, Lombok, bien ou pas bien ?

À la suite de cet argumentaire, il peut vous être difficile de choisir d'utiliser (ou conserver) Lombok ou non. Il est indéniable que cet outil présente de nombreuses qualités.

Certains avis négatifs sont liés à un manque de maîtrise du fonctionnement et de mauvaises expériences autour de Lombok. Dans ces situations, nous pensons qu'il est dommage de se priver de sa puissance alors qu'il est probablement moins coûteux d'investir sur la compréhension de l'outil et des bonnes pratiques associées.

Néanmoins, pour des projets reposant sur des écosystèmes techniques complexes, il peut être préférable de restreindre l'usage de Lombok à certaines classes (DTO, Value Objects) afin de limiter les mauvaises surprises.

Qu'on choisisse d'utiliser Lombok ou non, la décision devrait se faire collégialement et de façon éclairée.

Un grand merci à Antoine Alberti, Arnaud Courtès, Quentin Dorme, Laurent Gautho-Lapeyre, Julien Genty, Kevin Hantzen et Christian Spérandio, tous développeurs chez Arolla, pour les discussions endiablées qui ont mené à l'élaboration de cet article.

(3) L'oncle regretté de Peter Parker, aka Spiderman

Papier électronique avec PybStick26, Arduino ou Raspberry

Philippe MARTIN
Consultant Freelance
Formateur Python
Président de l'association LudikSciences
ludiksciences@gmail.com

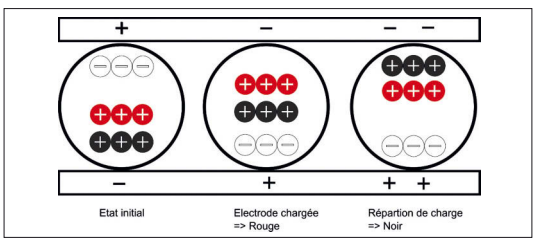
J'ai découvert le papier électronique (e-paper ou e-ink en anglais), il y a environ 4 ans avec l'idée d'utiliser ce support pour fabriquer des badges pour les salons (Maker Faire, Nante Maker Campus ou Makeme Fest) où mon association présente ses projets.

L'intérêt de ces écrans réside dans son excellente lisibilité même en plein soleil ainsi que son absence de consommation électrique une fois l'image affichée. Ce n'est pas pour rien que cette technologie est utilisée sur les liseuses numériques puisqu'elle permet de ne pas consommer de batterie pendant l'affichage d'une page.

Comment ça marche ?

Ces écrans sont basés sur l'affichage électrophorétique, je vous renvoie vers Wikipédia (ou tout autre site) pour plus de détails [1]. En synthèse, il s'agit d'une multitude de microcapsules contenant des particules blanches chargées négativement et des particules noires chargées positivement. En appliquant un champ électrique, on déplace des particules vers la surface visible de la microcapsule et les autres à l'opposé. En fonction de la polarité appliquée, un « pixel » est noir ou blanc.

Il existe également des papiers électroniques avec une couleur supplémentaire (en général Rouge ou Jaune).



Depuis 2016, certains écrans sont en mesure d'afficher 4096 couleurs via l'utilisation de 5 couleurs de particules (blanche, noir, cyan, magenta et jaune), un peu sur le modèle de l'impression en quadrichromie. Le point positif de ce système est qu'une fois les particules en place, elles restent en position même si les électrodes ne sont plus chargées. L'inconvénient de cette rémanence est qu'il est nécessaire pour rafraîchir l'écran de procéder à plusieurs inversions de polarité pour « ranger » toutes ces particules dans leur position initiale créant d'importants temps de rafraîchissement : plusieurs secondes. Waveshare fournit un tableau récapitulatif en fonction du modèle d'écran de 2s à 27s [2].

Choisir son écran

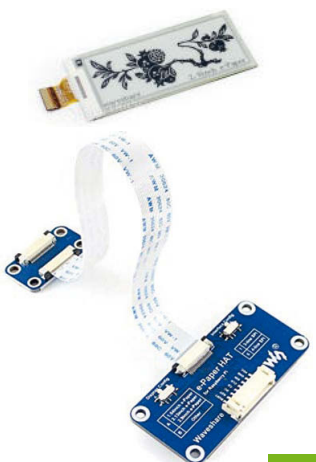
Les critères habituels d'un écran sont à prendre en considération (taille, résolution) avec bien entendu la partie budget qui est exponentielle (9 € pour notre écran d'environ 3" et on dépasse les 100 € pour un écran de 8" et +). L'ajout d'une couleur ou de plusieurs est séduisant, mais cela implique des temps de rafraîchissement dégradés (pour un écran 1,54 pouce, on passe de 2s à 8s pour obtenir, en plus du noir et blanc, la couleur rouge et on passe à 27s si on souhaite du jaune). Notons que les grands écrans (10" et +) conservent des taux de rafraîchissement plutôt corrects. Enfin, certains modèles embarquent le module de programmation ce qui est très confortable pour un montage pérenne, mais devient coûteux si l'écran est utilisé pour un affichage fixe comme dans notre cas de badge. Un dernier point, les plus anciennes générations ne proposent pas le rafraîchissement partiel, c'est-à-dire une zone sans avoir à traiter toute l'image, cela reste cruciale pour des applications où les valeurs affichées doivent régulièrement varier. Heureusement, Waveshare propose quand vous sélectionnez un écran sur leur boutique en ligne, en bas de page un tableau récapitulatif assez complet, il ne lui manque que les tarifs.

Selection Guide

PART NUMBER	COLORS	GREY SCALE	RESOLUTION	DISPLAY SIZE (MM)	OUTLINE DIMENSION (MM)	FULL REFRESH TIME (S)	PARTIAL REFRESH	FLEXIBLE	INTERFACE
1.02inch e-Paper	black, white	2	128×80	21.76 × 14.00	32.57 × 18.60 × 0.98	2	√		SPI
1.54inch e-Paper	black, white	2	200×200	27.60 × 27.60	37.32 × 31.80 × 1.05	2	√		SPI
1.54inch e-Paper (B)	red, black, white	2	200×200	27.60 × 27.60	37.32 × 31.80 × 1.05	8			SPI
1.54inch e-Paper (C)	yellow, black, white	2	152×152	27.51 × 27.51	37.30 × 31.80 × 0.98	27			SPI
2.13inch e-Paper	black, white	2	250×122	48.55 × 23.71	59.20 × 29.20 × 1.05	2	√		SPI
2.13inch e-Paper (B)	red, black, white	2	212×104	48.55 × 23.71	59.20 × 29.20 × 0.98	15			SPI

Matériel utilisé pour notre montage

- 1 carte PybStick26 environ 12-13 €
- 1 carte microSD de 16 Go classe 10
- 1 écran Waveshare 296x128, 2,9 pouces 9 € disponible sur waveshare.com
- 1 shield E-Paper Driver HAT Waveshare 9 € disponible sur waveshare.com



Le shield contient un régulateur de tension 5V -> 3V ainsi que des entrées/sorties permettant un fonctionnement avec une PybStick, un Arduino ou un Raspberry. Il est compatible avec les écrans de la marque et dispose de deux interrupteurs permettant de gérer les différents modèles. Il est livré avec un câble avec terminaison Dupont mâle et une rallonge permettant d'éloigner l'écran.

L'écran utilisé nécessite de positionner les 2 interrupteurs de la manière suivante : interrupteur Display config sur la position A / Interface Config sur 0 pour une liaison SPI 4 signaux :

- SCLK : Serial Clock, Horloge (généré par le maître).
- MOSI : Master Output, Slave Input (généré par le maître).
- MISO : Master Input, Slave Output (généré par l'esclave).
- SS : Slave Select, Actif à l'état bas (généré par le maître) et appelé ici CS

Sous Windows, j'utilise PuTTY [3] et Notepad++ [4] qu'il convient de configurer pour Python, à savoir éviter le mélange tabulation et espaces (Menu Paramètres/Préférences…).

Figure 1

La carte PybStick est reconnue comme une clef USB. S'il existe, une carte SD, le lecteur correspond à celle-ci sinon au 47ko de mémoire flash.

Les deux fichiers nécessaires à son bon fonctionnement sont :

- boot.py exécuté au démarrage de la carte et qui permet de configurer ses options si nécessaires.
- main.py qui contiendra le programme exécuté après boot.py

Pour se connecter à REPL sous Putty, il nous faut connaître le port COM utilisé, direction gestionnaire de périphériques : ici, il s'agit du port COM 13 (il suffit de débrancher/rebrancher la carte pour identifier le port qui disparaît/apparaît) **Figure 2**

À travers PuTTY : **Figure 3**

Cette fenêtre nous sera utile pour voir les retours de notre programme, tester une syntaxe ou rebooter la carte via la commande CTRL+D

Figure 1

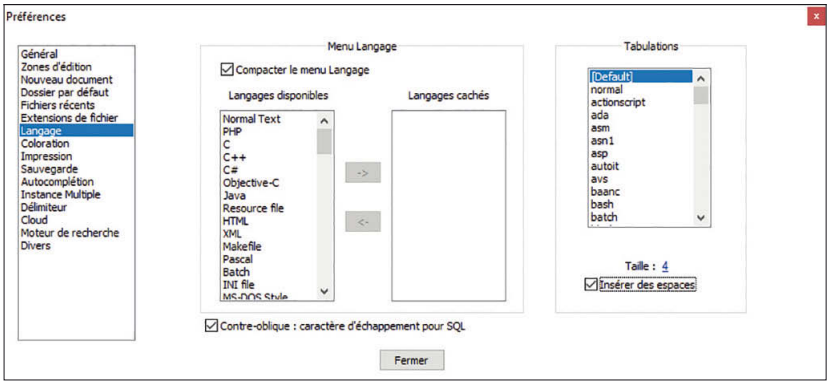


Figure 2

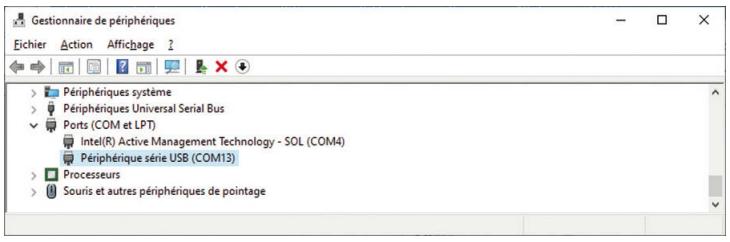
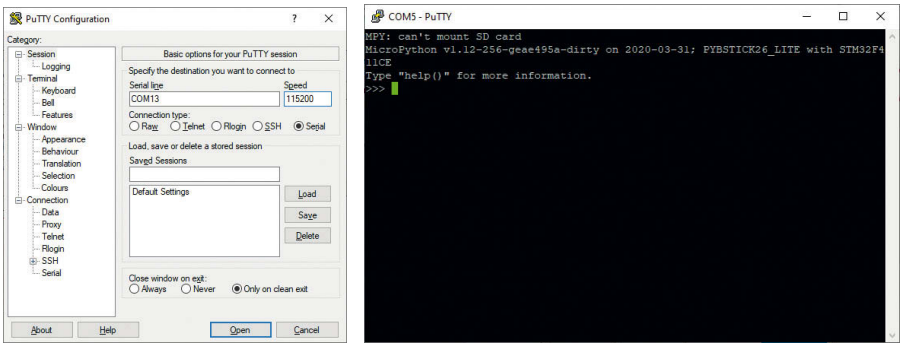
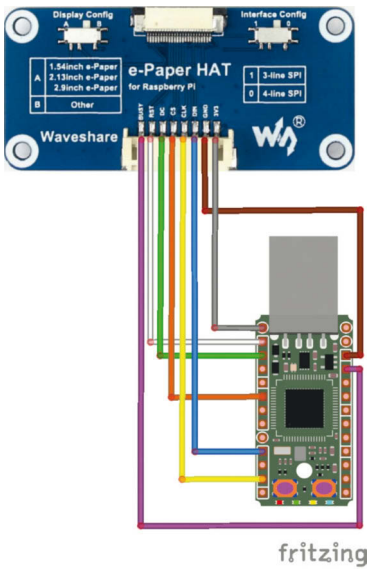


Figure 3



Montage

E –paper Driver HAT PybStick26	
VCC (+5V)	VIN
GND	GND
CS	S11
RESET (facultatif)	S3
DC	S5
SDI(MOSI)	S19
SCLK	S23
LED	3,3V
SDO(MISO)	S21



Principe de fonctionnement

Les pixels disposent de 2 états (Noir ou Blanc) qui peuvent être codés sur 1 bit : 0 pour noir et 1 pour blanc. Comme l'électronique de ce type d'écran attend un octet soit 8 bits donc 8 pixels, on affichera par groupe de 8 pixels (XX____X) avec deux pixels noirs au début et un en dernière position, nous devrons donc envoyer l'octet suivant : 00111110.

En Python, un binaire s'écrit avec le préfixe 0b, voici la représentation en hexadécimale de ces 8 pixels :

```
>> hex(0b00111110)
'0x3e'
```

Cette transformation sera utile pour l'affichage sur notre écran.

Affichage de nos premiers pixels

Comme notre écran fait 128 pixels de hauteur, cela nous fait 16 séries d'un octet pour remplir une colonne. Pour un quart de l'écran à partir du motif que nous avons vu précédemment (le motif sera répété 16 fois pour faire une ligne complète 128 = 16*8), il nous faudra envoyer (296/4)*16 = 1184 fois celui-ci.

Waveshare fournit des bibliothèques Python [5] et C pour chacun de ses écrans, dans notre cas, nous allons copier/coller la bibliothèque **epaper2in9.py** sur la PybStick.

Ensuite nous créons un nouveau fichier : zebre.py [6] :

```
import epaper2in9 # Bibliothèque ecran
from machine import Pin, SPI # Bibliothèque des entrées sorties

# Init PybStick26
spi = SPI(baudrate=24000000,miso=Pin("S21"), mosi=Pin("S19", Pin.OUT),
 sck=Pin("S23", Pin.OUT))
cs=Pin("S11");dc=Pin("S5");rst=Pin("S3");busy = Pin("S8")

e = epaper2in9.EPD(spi, cs, dc, rst, busy)
e.init()

w = 128;h = 296;x = 0;y = 0

#on vide l'écran
e.clear_frame_memory(b'\xFF')
image = ('\x3e'*1184).encode()

#On envoi l'image
e.set_frame_memory(image,x,y,w,h)
e.display_frame()
```

L'image est construite en créant un tableau hexadécimal en répétant 1184 fois le code hexadécimal de notre motif puis elle est transmise à l'écran via la méthode **set_frame_memory** qui prend en paramètres le tableau ainsi que la zone concernée (coordonnées x,y, largeur et longueur –dans notre cas, la totalité de l'écran)

Créer son badge

Vous trouverez diverses méthodes pour créer une chaîne hexadécimale correspondant à l'image sélectionnée. Ma méthode pour obtenir une image nette consiste à créer une image correspondant à la résolution de l'écran 296x128, j'utilise le logiciel vectorial Inkscape, mais Gimp, Illustrator ou Photoshop conviendraient également. J'exporte celle-ci en format BMP et utilise le logiciel image2lcd [6], créé par WuBoJian, pour obtenir un fichier contenant l'image convertie en hexadécimale. **Figure 4**

Dans image2lcd, il convient de sélectionner la valeur : C Array pour Output type file, Scan Mode : Vertical Scan sans oublier d'inverser les couleurs puis un clique sur Save et vous vous retrouvez avec un fichier TESTU.c qui contient une chaîne hexadécimale, en C. La conversion en Python va consister à supprimer la première ligne puis les caractères de fin de ligne « \n » et « \r » et remplacer les chaînes « ,0X » par « \x ». Enfin, on remplacera le premier « 0X » par « b'\x » sans oublier de fermer la chaîne de caractère en fin de fichier. Je vous ai mis un script Python appelé ConvEpaper.py [6] qui s'occupera de réaliser ses étapes et de créer un fichier résultat TESTU.py. Il vous suffira ensuite d'importer dans votre code via la commande **import testu** pour disposer d'une variable image contenant votre réalisation. **Figure 5**

```
import epaper2in9 # Bibliothèque ecran
from testu import image # Image générée via image2lcd
from machine import Pin, SPI # Bibliothèque de gestion des
entrées sorties

# Init PybStick26
spi = SPI(baudrate=24000000,miso=Pin("S21"), mosi=Pin("S19",
 Pin.OUT), sck=Pin("S23", Pin.OUT))
cs=Pin("S11");dc=Pin("S5");rst=Pin("S3");busy = Pin("S8")

e = epaper2in9.EPD(spi, cs, dc, rst, busy)
e.init()

w = 128;h = 296;x = 0;y = 0

#on vide l'écran
e.clear_frame_memory(b'\xFF')
e.set_frame_memory(image,x,y,w,h)

e.display_frame()
```

Le code final est quasi identique à notre motif zébré, il se différencie par le déport de la variable image dans un module externe pour gagner en efficacité puisqu'en cas de changement d'image, le corps du programme pourra rester identique.

Conclusion

Ce type d'écran reste cher et les temps de rafraîchissement ne permettent qu'un usage statique. À l'opposé sa très faible consommation et son excellente lisibilité au soleil en font un choix très opportun pour divers usages. Par exemple, en couplant ce type d'écran à un ESP32 et son mode deep sleep ne consommant qu'une dizaine de mA [7], on peut imaginer un montage sur batterie pour effectuer des mesures de capteurs plusieurs fois par jour et utiliser l'écran pour avoir un état de la batterie ainsi que les dernières mesures enregistrées. Pour afficher du texte, vous pourrez utiliser des générateurs de polices qui transformeront vos textes en images [8].

[1] https://fr.wikipedia.org/wiki/Papier_%C3%A9lectronique
[2] https://www.waveshare.com/e-paper-driver-hat.htm
[3] https://www.putty.org/
[4] https://notepad-plus-plus.org/
[5] https://github.com/waveshare/e-Paper/tree/master/RaspberryPi_JetsonNano/python/lib/waveshare_epd
[6] https://github.com/philippeminerve/PYBStick26/tree/master/micropython/Epaper
[7] https://letmeknow.fr/shop/fr/blog/142-tutoriel-les-sleep-modes-de-lesp32
[8] https://github.com/mchobby/freetype-generator

Figure 4

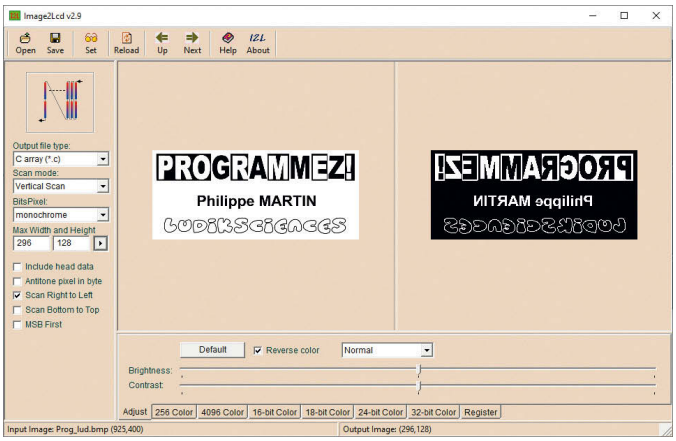


Figure 5

Sallah KOKAINA

Ingénieur en informatique et auteur du livre « Software Craftsmanship : L'art du code et de l'agilité technique en entreprise » (Éditions ENI 2019). Sur ces 12 dernières années, il a tenu différents rôles (Développeur, CTO, Qualiticien, Manager Technique…) au sein de différentes structures et y a mis en place différentes solutions basées notamment sur Scala, Kotlin et Java. Twitter (@koksbox).

Kotlin – 360 sur un écosystème full-stack et pro-entreprise

PARTIE 1

Ces dernières années, Kotlin n'a cessé de gagner en popularité et en adoption autant par les communautés de développeurs que les entreprises. Il détient à cet effet de nombreux atouts qui en font un challenger redoutable pour les stacks JVM concurrentes, tels que Java, Scala et Clojure. En effet, il dispose d'une interopérabilité totale avec Java, il est depuis 2019 le langage officiel de programmation Android et vient avec un écosystème de librairie en faisant une référence fullstack crédible.

En regardant les benchmarks des dernières années, Kotlin se classe systématiquement parmi le top 10 des langages, avec une croissance continue année après année, et ce depuis 2017. Ainsi, Kotlin est l'un des langages avec l'une des communautés d'adeptes qui compte la plus grande progression, passant de 1,1 million en 2017 à 2,3 millions en 2020(1).

Et ce n'est pas surprenant. Kotlin a un très bon niveau d'outillage de support créé par les fournisseurs d'IDE, à l'image de Jetbrains. De plus, il vient avec un DSL expérimental depuis sa version 1.3 et qui permet la gestion de contrat code. Cela offre ainsi aux développeurs une garantie sur le comportement du code tout en allégeant la charge sur le compilateur(2).

Par ailleurs, son interopérabilité avec Java en fait un langage pro-JVM de plus en plus apprécié par les JAVAistes. Il est utile de noter que les innovations syntaxiques introduites par Kotlin se retrouvent de plus en plus intégrées dans les versions récentes de Kotlin. D'ailleurs, sur les dernières tendances, Kotlin dépasse désormais Scala et Closure, prenant ainsi la 2e place en tant que langage JVM après Java. Sans compter, qu'avec l'adoption de Kotlin dans des frameworks comme Sprint Boot ou JHipster vient en plus pour faciliter la création de solution production-ready.

En plus de l'engouement communautaire, on peut noter quelques faits remarquables qui expliquent la popularité croissante de l'écosystème Kotlin dans l'industrie et les cursus scolaires:

- Kotlin est adopté par des entreprises de renoms et de toutes tailles : Twitter, Reddit, Pinterest, Uber, Coursera, Evernote, Slack, IceRock, Careem, et Trello pour en citer quelques-unes.
- Les offres d'emplois autour de Kotlin ont augmenté de plus de 1400% depuis 2017. Source: Dice.
- Kotlin est l'un des langages de programmation connaissant la plus forte progression, atteignant la 4e place dans le rapport Github Octoverse 2019.
- PYPL place Kotlin à la 12e place des langages les plus populaires, avec une forte tendance à la hausse en 2020.
- D'après le rapport 2019 StackOverflow Developer Survey, c'est le 4e langage le plus aimé et recherché par les communautés.
- Au moins 82 universités dans le "Times Higher Éducation Rankings 2020" utilisent Kotlin pour enseigner le développement Mobile, Orienté objets et fonctionnel, Design Patters, Programmation parallèle et concurrente, ainsi que bien d'autres cours (Source: internal Teaching Kotlin Study).
- 22 du top des 100 universités dans "THE Rankings 2020" intègrent Kotlin

Dans cet article, on va s'intéresser à l'écosystème Kotlin qui justifie cette popularité montante et en fait un langage doté d'un arsenal complet pouvant plaire tant aux entreprises qu'aux passionnés de développement.

Un écosystème full stack

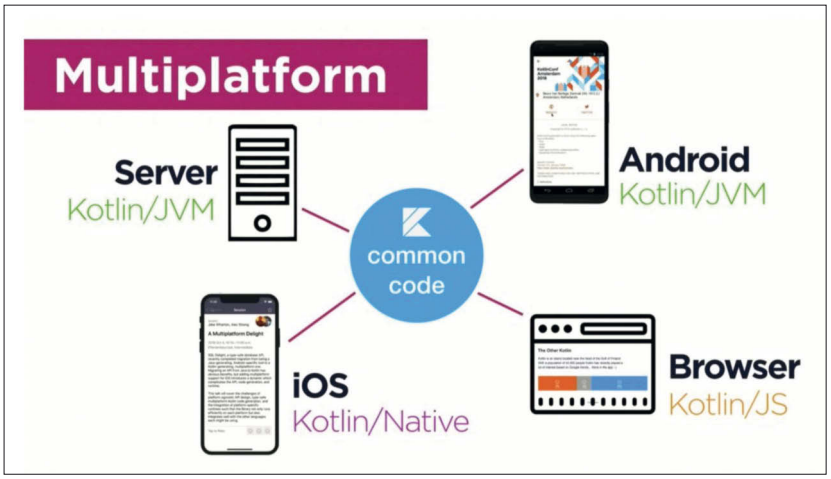
Avec Kotlin, il est possible de construire tant des frontends applicatifs web ou mobiles que des backends microservices, serverless. On peut ainsi avoir de bout en bout des solutions se basant principalement sur Kotlin. Cerise sur le gâteau, en plus d'avoir un langage commun pour simplifier la collaboration, la gestion de compétence et la maintenance applicative entre couches, on dispose de :

- Un riche écosystème de frameworks de gamme entreprise.
- Une alternative fiable pour construire de solutions modernes et robustes.
- Une technologie aux contributions communautaires de plus en plus actives.

1. Kotlin multiplatform (KMP). **Figure 1**

Avec l'arrivée Kotlin/Native, il est désormais possible de compiler du code Kotlin en binaires natifs. Ainsi, on peut compiler une base de code Kotlin en une librairie Objective-C intégrable dans un projet iOS comme on le ferait pour toute autre librairie native. Le code produit ainsi peut s'exécuter en

Figure 1



natif, sans la nécessité d'une machine virtuelle intermédiaire pour interpréter le code au runtime. Par exemple, React-Native se base sur JavaScriptCore, qui est le moteur JavaScript derrière Safari. En plus, il utilise une autre couche dénommée React Native Bridge qui s'assure de la communication entre le runtime natif et thread JavaScript. Avec Kotlin/Native, on l'aura compris, il n'y a plus d'intermédiaires, ce qui fait de Kotlin Multiplatform un nouveau challenger ouvrant la porte de l'IoT à l'écosystème Kotlin.

a. Un nouveau challenger

React-Native (RN) a été pendant longtemps, et reste à ce jour, la référence en développement d'applications mobiles multi-plates-formes. Avec une base de code commune, écrite en JavaScript, vous pouvez générer le binaire applicatif pour les plateformes Android (apk) et iOS (ipa).

C'est ce qu'on appelle, faire d'une pierre deux coups. Toutefois, et contrairement à du développement purement natif (Swift, Kotlin), ce framework de développement vient avec quelques inconvénients en termes de : Sécurité, Performance, Expérience utilisateur.

- **Sécurité**: RN, comme tout framework JavaScript-based, est sujet à différentes vulnérabilités (XSS, ACE, Zil Slip, SSR attacks).
- **Performance**: Comparé à un mode natif, RN vient avec la limitation JS vu qu'il fonctionne par défaut en single thread.
- **Expérience utilisateur**: Il est possible d'obtenir un look and feel similaire à celui d'une expérience native, toutefois on peut rencontrer certaines limitations quand il s'agit de créer des interfaces complexes, notamment en terme de navigation.
- **Expérience développeur**: Bien que le développement initial est très rapide, on se rend compte très vite qu'il est nécessaire d'avoir aussi une connaissance des technologies natives (Kotlin, Swift). Et cela, que ce soit pour debugger ou pour étendre les capacités des projets natifs sous-jacents afin d'exposer des libraires natives sur la base de code RN.

C'est là que Kotlin Multiplaform, depuis la sortie de son passage en version Alpha en août 2020, offre une alternative particulièrement intéressante. Sachant qu'à ce jour, et au-delà de la version globale, différents sous-composants clés sont déjà en Beta avec la brique Kotlin/JVM désormais en version stable : **figure 2**.

Et pour cause, sur les dimensions citées précédemment, on peut apprécier quelques avantages :

- **Sécurité** : Le passage par des SDK natives fournit une sécurité naturellement plus élevée, bénéficiant des derniers patch d'OS, et rendant les solutions produites plus fiables notamment en terme de protection de données.
- **Performance** : On est dans des performances égales au natif, avec des librairies telles que kotlinx-coroutines, il est facile de partager des routines d'algorithmes multi-thread s'intégrant naturellement aux capacités hardware des appareils mobiles.
- **Expérience utilisateur - UX**: KMM est principalement utilisé pour le partage de logique métier, et bien qu'il soit possible de partager également l'interface utilisateur avec une approche déclarative (D-KMP), au final peut aussi isoler les interfaces et les implémenter directement en natif, bénéficiant de la force des IDE (Android Studio, Xcode Studio), des approches trendy (SwiftUI, Jetpack Compose), pour ainsi créer des expériences totalement natives.
- **Expérience développeur - DX**: le Debugging est bien plus efficace vu qu'on peut utiliser directement les IDE natifs.

Component	Stability level
Kotlin/JVM	Stable
Kotlin/Native Runtime	Beta
KLib binaries	Alpha
expect/actual language feature	Beta
Multiplatform Gradle plugin	Beta
Kotlin/Native interop with C and Objective C	Beta
CocoaPods integration	Beta

Figure 2

Cette alternative a reçu l'attention de plusieurs géants de l'informatique et a été à l'origine de quelques success stories(14) : Careem, Netflix, Philips, Autodesk, VMware, Cash App.

Kotlin multiplatform est un nouveau challenger dans le monde du développement multiplatform. Il offre une alternative intéressante permettant de concilier productivité et maturité. Une tendance technologique à suivre de très près sur ces prochains mois.

b. Des frameworks vitaux

Bien qu'en étant à sa version alpha, en plus d'un support et sponsorship fort de Jetbrains, la communauté open source autour de Kotlin multiplatform est très active. On y trouve quelques frameworks clés à garder dans sa boîte à outils de développeurs. La liste des frameworks[5] ci-dessous est une sélection de briques populaires et pertinentes pour la création de solutions KMM.

- **SQLDelight** - Génère des API de type sécurisé à partir des instructions SQL, partageables entre iOS et Android. Il vérifie au moment de la compilation le schéma SQL, les instructions et les migrations et fournit des fonctionnalités IDE telles que la saisie semi-automatique et le refactoring qui en simplifie l'écriture et la maintenance. Voici là un cache SQL commun entre ces deux environnements.
- **Ktor** - Mutualiser la couche networking avec ce framework kotlin complet. En plus d'un support cilent pour l'accès aux API Rest, Ktor permet aussi la création de serveurs applicatifs, voire microservices.
- **Multiplatform Settings** - Bibliothèque pour les applications mobiles multi-plates-formes, afin que le code commun puisse conserver les données clé-valeur. Il stocke les éléments en utilisant SharedPreferences sur Android et NSUserDefaults sur iOS.
- **Suparnatural-cache** - Un cache en mémoire ultra-rapide et thread-safe avec des schémas de hachage configurables, soutenus par des repository de persistance avec des I/O bloquantes/non-bloquantes.
- **Hydra Log** - Permet d'écrire des logs dans le module commun. La façon dont les logs seront écrits peut aussi

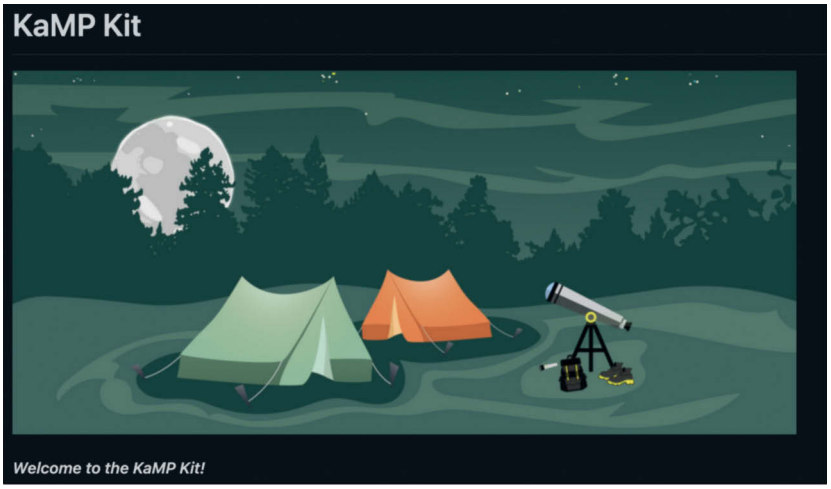


Figure 3

être définie indépendamment pour chaque plateforme.

- **Koin** - Une bibliothèque d'injection Kotlin intelligente pour permettre de rester concentré sur la rédaction du code de l'application et mutualiser un maximum de composant: ViewModels, Commandes, Modèles..
- **Mokko-MVVM** - Une brique Model-View-ViewModel, signée IceRockDev, permettant de partager les ViewModels entre Android et iOS. En partageant la logique business, on peut arriver à partager presque 80% du code entre les projets iOS et Android.
- **Mokko-resources** - Partager les ressources entre les plateformes mobiles android & iOS : fonts, images, values...

c. Un kit pour bien démarrer (KampKit)

Démarrer du bon pied avec un code production-ready, c'est ce qu'il y a de plus dur à réaliser avec Kotlin Multiplatform. Mais pas de panique, il y a KampKit ! **Figure 3**.

KampKit est un starterkit crée par Touchlab et qui combine plusieurs briques open source pro-KMP. Avec ce boilerplate, il devient facile de démarrer rapidement, en voyant comment les concepts clés sont mis en œuvre, bénéficiant d'une base de code presque production-ready :

- Injection de dépendance avec Koin
- Appel REST avec Ktor
- Gestion de log avec Kermit
- Cache SQL avec SQLDelight

À l'image de create-react-app pour React, ce Kit de développement a été conçu pour faciliter une entrée en matière rapide. En plus, il intègre des ressources et pointeurs pour monter en compréhension sur certains concepts fondamentaux: gestion d'appels concurrents, debugging, gestion de build.

2. Kotlin.JS

Voici une autre perle qui contribue à l'arsenal Kotlin et qui s'intègre naturellement à Kotlin multiplaform. Pour cela, la stack s'appuie sur le compilateur Kotlin/JS IR qui vient avec quelques atouts. Le compilateur Kotlin/JS optimize la taille du js généré en éliminant le code mort. Il génère aussi des fichiers de déclaration Typescript (d.ts), chose qui rend aisée la création d'applications hybrides mixant du code Typescript et Kotlin, et cela, tout en partageant du code commun via la couche Kotlin multiplatform.

Au-delà, Kotlin.js vient avec quelques frameworks intéressants qui boostent la productivité tout en s'intégrant de façon naturelle au reste de l'écosystème :

- **KVision**(7) est un framework web orienté objet qui permet d'écrire des applications en Kotlin/JS avec des composants prêts à l'emploi. Ils peuvent être utilisés comme blocs de construction pour l'interface utilisateur. Il offre la possibilité d'utiliser des modèles de programmation réactifs et impératifs pour créer les interfaces. On peut aussi utiliser des connecteurs pour Ktor, Spring Boot et d'autres frameworks pour l'intégrer aux applications côté serveur et partager du code à l'aide de Kotlin Multiplatform.
- **fritz2**(8) est un framework autonome pour la création d'interfaces utilisateur web réactives. Il fournit son propre DSL de type sécurisé pour la construction et le rendu des éléments HTML, et il utilise les coroutines et les flows de Kotlin pour exprimer les composants et leurs liaisons de données. Il fournit la gestion d'état, la validation, le routage et plus encore, et s'intègre aux projets Kotlin Multiplatform.
- **Doodle**(9) est un framework d'interface utilisateur vectoriel pour Kotlin/JS. Les applications Doodle utilisent les capacités graphiques du navigateur pour dessiner des interfaces utilisateur au lieu de s'appuyer sur le DOM, du CSS ou JavaScript. En utilisant cette approche, Doodle donne un contrôle précis sur le rendu des éléments arbitraires de l'interface utilisateur, des formes vectorielles, des gradients et des visualisations personnalisées.
- **Compose for Web**(10) apporte la boîte à outils Jetpack Compose UI de Google au navigateur. Il permet de créer des interfaces utilisateur web réactives en utilisant les concepts introduits par "Jetpack Compose". Il fournit une API DOM pour décrire le site web, ainsi qu'un ensemble expérimental de primitives de mise en page multi-plates-formes. "Compose for web" offre également la possibilité de partager des parties de code d'interface utilisateur et de logique sur Android, Desktop et web.

Alors, n'attendez plus, c'est le moment de vous y essayer à l'aide de lab hans-on, et notamment construire une web App Full stack à l'aide de KMM, pour cela: rendez-vous sur le site de KotlinLang(6).

3. Microservices – SpringBoot(11)

Pour les Javaistes, Springboot est la référence en tant que Framework de développement production-ready pour les backend web et plus récemment pour la réalisation de "heavy-weighted" microservices. Seul écueil, le nombre de boilerplate (setters, getters, hashcode…) qui pour le meilleur des cas nécessite l'intégration de librairies tierces telles que lombok. Imaginez un monde sans librairie tierce avec toute la puissance de Springboot, facilitant la gestion de tests et avec tous ces boilerplates en moins. Avec Kotlin, ce monde est là, avec une syntaxe légère et agréable :

Un class main plus fine

```
@SpringBootApplication class BlogApplication

fun main(args: Array<String>) {
 runApplication<BlogApplication>(*args)
}
```

Des controllers plus fin

```
@RestController
@RequestMapping("/api/user")
class UserController(private val repository: UserRepository) {
 @GetMapping("/")
 fun findAll() = repository.findAll()

 @GetMapping("/{login}")
 fun findOne(@PathVariable login: String) = repository.findByLogin(login)
}
```

Une JPA plus compacte

```
@Entity class User(
  var login: String,
  var firstname: String,
  var lastname: String,
  var description: String? = null,
  @Id @GeneratedValue var id: Long? = null)

Interface UserRepository : CrudRepository<User, Long> {
 fun findByLogin(login: String): User?
}

@DataJpaTest class RepositoriesTests @Autowired constructor(
  val entityManager: TestEntityManager,
  val userRepository: UserRepository) {

 @Test
 fun `When findByLogin then return User`() {
  val juergen = User("springjuergen", "Juergen", "Hoeller")
  entityManager.persist(juergen)
  entityManager.flush()
  val user = userRepository.findByLogin(juergen.login)
  assertThat(user).isEqualTo(juergen)
 }
}
```

Une gestion de test d'intégration plus légère

```
@WebMvcTest
class HttpControllersTests(@Autowired val mockMvc: MockMvc) {

 @MockkBean
 private lateinit var userRepository: UserRepository

 @Test
 fun `List users`() {
  val juergen = User("springjuergen", "Juergen", "Hoeller")
  val smaldini = User("smaldini", "Stéphane", "Maldini")
  every { userRepository.findAll() } returns listOf(juergen, smaldini)
  mockMvc.perform(get("/api/user/").accept(MediaType.APPLICATION
_JSON))
     .andExpect(status().isOk)
    .andExpect(content().contentType(MediaType.APPLICATION_JSON))
     .andExpect(jsonPath("\\$.[0].login").value(juergen.login))
     .andExpect(jsonPath("\\$.[1].login").value(smaldini.login))
 }
}
```

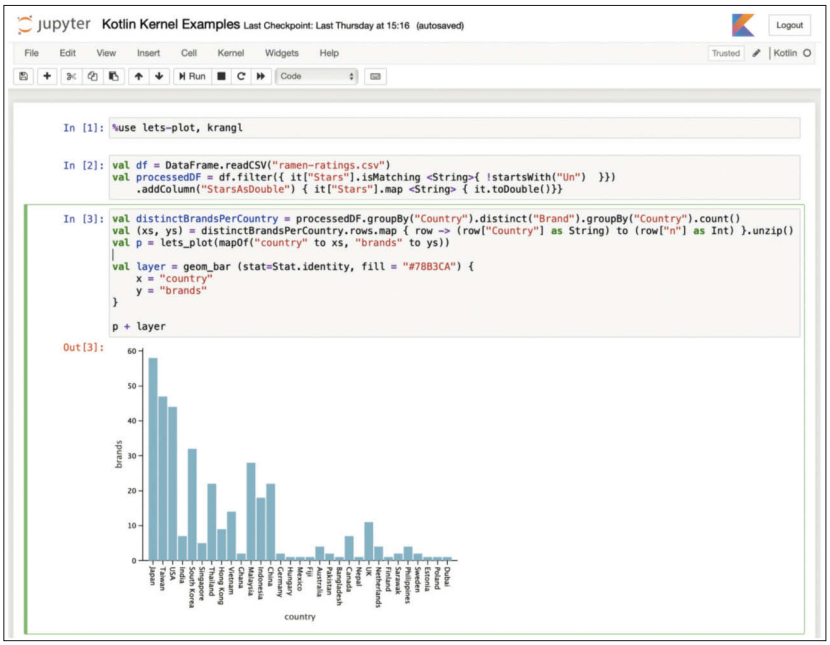


Figure 4

4. Data science

Voici la cerise sur le gâteau, eh oui, on peut construire des solutions Data Science fiables et avancées en dehors de Python. Et tout cela, en restant dans l'écosystème Kotlin.

Kotlin est pertinent pour la création de pipeline de traitement Data pour la production de modèle de Machine Learning avancés. Et si vous êtes un adepte de Juptyter en python, alors pas de dépaysement à avoir, Kotlin kernel offre la même expérience permettant de créer et partager des notebooks contenant du code, des graphes, markdowns et tableaux pro-IA. **Figure 4**.

En plus, et pour de moins en moins envier à Python, l'écosystème de librairies Kotlin créées par la communauté ne cesse de s'élargir avec une sélection s'appuyant en plus sur l'écosystème Java. Souvenez-vous, Kotlin est totalement interopérable avec Java.

- **Multik** : Une implémentation de tableaux multidimensionnels dans Kotlin. La bibliothèque fournit une API Kotlin-idiomatic, de type et de dimension safe pour les opérations mathématiques sur des tableaux multidimensionnels. Multik propose une JVM échangeable et des moteurs de calcul natifs, ainsi qu'une combinaison des deux pour des performances optimales.
- **KotlinDL** est une API de Deep Learning de haut niveau écrit en Kotlin et inspirée de Keras. Il propose des API simples pour former des modèles d'apprentissage en profondeur à partir de zéro. On peut importer des modèles Keras existants pour l'inférence et tirer parti de l'apprentissage par transfert pour adapter des modèles pré-entrainés.
- **kmath** est une bibliothèque inspirée de NumPy. Cette bibliothèque prend en charge les structures et opérations algébriques, les structures de type tableau, les expressions mathématiques, les histogrammes, les opérations de diffusion en continu, un wrapper autour de commons-math et koma, et plus encore.
- **krangl** est une bibliothèque inspirée du dplyr (R) et de pandas (Python). Cette bibliothèque fournit des

fonctionnalités pour la manipulation de données à l'aide d'une API de style fonctionnel ; elle comprend également des fonctions de filtrage, de transformation, d'agrégation et de remodelage des données tabulaires.

- **londogard-nlp-toolkit** est une bibliothèque qui fournit des utilitaires pour le traitement du langage naturel (NLP), tels que les intégrations de mots/sous-mots/phrases, les fréquences de mots, les mots d'arrêt, et bien plus.
- **Weka**- une collection d'algorithmes d'apprentissage pour les tâches de data-mining.
- Smile un système complet d'apprentissage automatique, de traitement du langage naturel, d'algèbre linéaire, de graphique, d'interpolation et de visualisation. Outre l'API Java, Smile fournit également une API Kotlin fonctionnelle avec Scala et l'API Clojure.
- **Apache Zeppelin** est une solution web populaire, similaire à Jupyter, pour l'analyse de données interactive. Elle fournit un support solide pour Apache Spark, ce qui est particulièrement utile pour l'ingestion de données. À partir de la version 0.9.0, Apache Zeppelin est livré avec un interpréteur Kotlin.

Et bien plus !

Unit Testing en Kotlin- Les tests unitaires dans Kotlin sont à la fois amusants et délicats. Nous pouvons beaucoup bénéficier des forces syntaxiques de Kotlin pour écrire des tests unitaires lisibles et concis. Mais pour écrire du code de test Kotlin idiomatique en premier lieu, une certaine configuration de test est requise. L'article de Philipp Hauer[13] contient les meilleures pratiques et les directives pour écrire des tests unitaire dans Kotlin qui sont idiomatiques, lisibles, concis et bien structurés.

Production-ready avec KHispster - https://khipster.dev/ - Un atout majeur de productivité en déclinaison de Jhipster. On peut ainsi configurer une application web Full Stack moderne en exécutant une seule commande (khipster).

```
npm i -g generator-jhipster-kotlin && mkdir kt-app && cd kt-app
khipster
```

En plus, on peut bénéficier d'un IDE en ligne qui permet de déclarer les entités, dépendances et caractéristiques du projet (db, auth, cache, kubernetes.), et ainsi puis générer un projet full-stack contenant des tests et la configuration de build multi-environement (dev, production) **Figure 5**.

Compose for desktop - Compose for Desktop simplifie et accélère le développement de l'interface utilisateur pour les applications de bureau et permet un partage étendu du code de l'interface utilisateur entre Android et les applications de bureau. Actuellement en alpha.

Podcast - https://talkingkotlin.com/ - Talking Kotlin est un podcast sur Kotlin qui aborde fréquemment les mises à jour apportées au langage, expliquant d'un épisode à l'autre comment certains concepts gagnent concrètement en maturité et adoption par tant les entreprises que les communautés de développement.

Conférence - KotlinConf a affiché complet pendant trois années consécutives en 2019, avec plus de 1700 participants. C'est un événement international à ne pas manquer pour rester à jour sur les tendances et les retours d'expérience.

Influenceurs - En complément, et pour finir, voici une liste d'influenceurs qu'il est intéressant de suivre, vu qu'ils façonnent l'écosystème de par leurs contributions et interventions communautaires :

- **Marcin Moskala** - le créateur de Kot. Academy et l'auteur du livre 'Android Development with Kotlin'. Il est également un développeur Android expérimenté passionné par Kotlin depuis sa première version bêta. Il entretient un blog dont la mission est de simplifier l'apprentissage de Kotlin.
- **Kevin Galligan** - Kevin a fondé Touchlab en 2010 et code professionnellement depuis plus de 20 ans. Il a été un pionnier du mobile, d'abord sur Android et aujourd'hui à la pointe de KMM. Il participe et anime la réunion des développeurs Android de NY et Droidcon NYC.
- **Alex Mikhailov** - Chief Technology Officer at IceRockDev, une startup qui contribue fortement à l'open source autour de KMM, notamment avec la suite de librairies MOKO (https://github.com/icerockdev).
- **Antonio Leiva** - Blog régulièrement sur Kotlin en fournissant des tips pour booster votre productivité dans l'implémentation de solution Android.
- **Andrey Breslav** a été depuis 2010 en Lead sur le design et l'implémentation de Kotlin au de JetBrains. On pourra apprécier en ligne certaines de ses interventions comparant Java à Kotlin.
- **Brady Aiello** - principal contributeur du projet starter kit KMM: KampKit.
- **Sebastian Aigner** - Kotlin JS évangéliste et Developer Advocate.

Figure 5

Références :

(1) https://dev.to/deveconomics/infographic-programming-languages-adoption-trends-2020-41b
(2) https://medium.com/analytics-vidhya/top-programming-languages-trends-in-2020-which-one-you-should-learn-dd1bc6a529bd
(3) https://snyk.io/blog/kotlin-overtakes-scala-and-clojure-to-become-the-2nd-most-popular-language-on-the-jvm/
(4) https://kotlinlang.org/education/why-teach-kotlin.html
(5) https://github.com/bipinvaylu/awesome-kotlin-multiplatform
(6) https://kotlinlang.org/docs/js-overview.html#get-started-with-kotlin-js
(7) https://kvision.io
(8) https://www.fritz2.dev
(9) https://nacular.github.io/doodle/
(10) https://jb.gg/compose-web
(11) https://spring.io/guides/tutorials/spring-boot-kotlin/
(12) https://kotlinlang.org/docs/data-science-overview.html
(13) https://phauer.com/2018/best-practices-unit-testing-kotlin/
(14) https://kotlinlang.org/lp/mobile/case-studies/

Philips PerfectCare série 9000 :
une centrale vapeur boostée au deep learning !

Dans les n°244 & 246, nous avons présenté les principes du deep learning en détaillant le fonctionnement des réseaux de neurones de type perceptron multicouche et convolutif. Ces technologies ne sont pas réservées aux puissants serveurs, mais peuvent également être utilisées pour « donner de l'intelligence » à des produits du quotidien. Ainsi la nouvelle centrale vapeur PerfectCare série 9000 de Philips intègre un réseau à convolution pour reconnaître en temps réel la nature du tissu repassé !

Jean-Christophe Riat

Passionné depuis le lycée par l'informatique, j'enseigne l'intelligence artificielle depuis plus de 20 ans à l'EPSI Paris, 1re école d'informatique créée en France par des professionnels [7]. Je suis enthousiasmé par les progrès récents en apprentissage machine, d'autant qu'internet rend accessibles à tous les ressources (tutoriels, outils …) pour comprendre et mettre en œuvre les techniques de « Deep Learning ».

Annonce récente de Philips

Philips commercialise depuis quelques mois une nouvelle centrale vapeur baptisée PerfectCare Série 9000 [1]. Il s'agit d'un produit haut de gamme, destiné au grand public, avec un prix proche des 700 €. Ce positionnement se justifie par un produit extrêmement innovant, par rapport à la concurrence, avec l'utilisation d'une caméra couplée à de l'intelligence artificielle pour reconnaître les tissus et adapter automatiquement la température et la quantité de vapeur (cf. **encadré n°1**).

Comme le montre le film publicitaire [2], l'objectif est d'améliorer l'expérience utilisateur, en simplifiant l'usage du produit. Il n'est plus nécessaire de procéder à des réglages manuels : les boutons pour sélectionner le type de tissus (laine, lin, coton…) ou le caractère délicat du textile ont disparu ! Ils sont remplacés par une caméra, couplée à un microcontrôleur, qui automatise les réglages en temps réel. La vidéo [3] détaille ce fonctionnement, avec différents exemples.

Recherche d'informations grâce aux brevets

Pour valoriser son produit, Philips met en avant l'utilisation de l'intelligence artificielle, mais ne donne aucune précision, ni sur la technique utilisée ni sur sa mise en œuvre. Une piste intéressante, pour essayer d'en savoir plus, consiste à faire une recherche sur les brevets. Pour se protéger des contrefaçons, les industriels ont souvent recours à la protection intellectuelle [4]. Un brevet est une sorte de titre de propriété sur une solution technique, qui interdit l'usage à titre commercial du dispositif protégé, pour toute autre entreprise que la société détentrice.

Mais il faut savoir qu'un brevet ne reste confidentiel que pendant une durée limitée, car il est rendu public 18 mois après le premier dépôt : la solution technique reste protégée, mais peut être connue par les concurrents… et par toute personne curieuse ! En effet, les brevets sont gratuitement accessibles sur internet, par exemple avec le moteur de recherche « patents.google » [5]. Avec une recherche sur la société dépositaire et des mots clés associés au type d'application, on trouve facilement les brevets déposés par Philips pour sa centrale vapeur. Le plus intéressant porte la référence « WO2020254157 » (cf. **encadré n°2**), avec un dépôt au niveau mondial, qui couvre plus de 150 pays.

Principes de fonctionnement du système

La lecture d'un brevet n'est pas forcément aisée, car il s'agit d'un document à la fois technique et juridique, sans objectif pédagogique. Malgré tout, avec l'utilisation des figures toujours présentes après le texte, il est généralement possible pour une personne familière du domaine technique considéré de comprendre le fonctionnement protégé.

Pour la centrale vapeur, nous avons construit, à partir des informations disponibles, le synoptique représenté dans l'**encadré n°3**. Le système utilise deux capteurs en entrée : une caméra pour acquérir des images du textile et un capteur de mouvement. Ce dernier est principalement utilisé pour gérer les situations particulières où le fer à repasser reste immobile, avec des logiques de diminution automatique de la température de la semelle pour réduire la consommation électrique et éviter de brûler le linge en cas de contact permanent trop prolongé. On remarque qu'il n'y a plus de consignes de réglage par l'utilisateur, qui a uniquement la possibilité d'activer ou non le mode « auto » pour la gestion automatique de la vapeur.

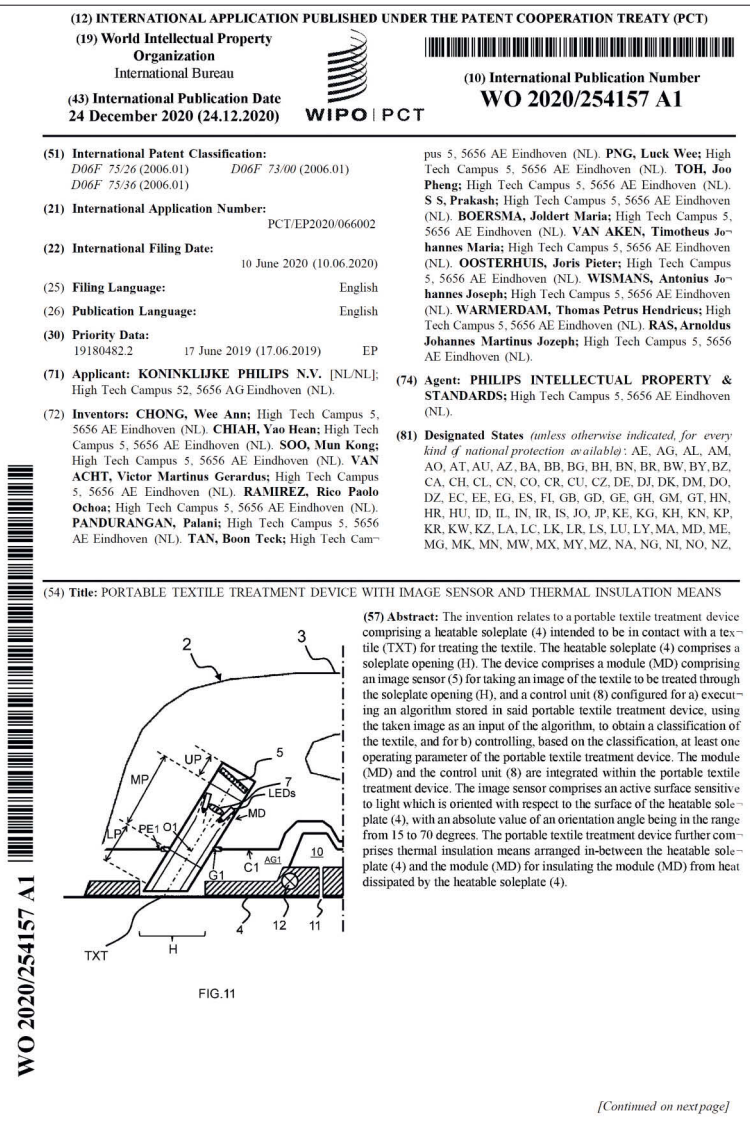
Encadré n°1

PLAQUETTE COMMERCIALE DE LA CENTRALE VAPEUR

Valorisation du produit par Philips avec l'annonce de l'utilisation de l'intelligence artificielle pour reconnaître les tissus et adapter automatiquement la température et la quantité de vapeur.

Encadré n°2

BREVET MONDE DÉPOSÉ PAR LA SOCIÉTÉ PHILIPS



(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 December 2020 (24.12.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/254157 A1

(51) International Patent Classification:
D06F 75/26 (2006.01) *D06F 73/00* (2006.01)
D06F 75/36 (2006.01)

(21) International Application Number:
PCT/EP2020/066002

(22) International Filing Date:
10 June 2020 (10.06.2020)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
19180482.2 17 June 2019 (17.06.2019) EP

(71) Applicant: **KONINKLIJKE PHILIPS N.V.** [NL/NL]; High Tech Campus 52, 5656 AG Eindhoven (NL).

(72) Inventors: **CHONG, Wee Ann;** High Tech Campus 5, 5656 AE Eindhoven (NL). **CHIAH, Yao Hean;** High Tech Campus 5, 5656 AE Eindhoven (NL). **SOO, Mun Kong;** High Tech Campus 5, 5656 AE Eindhoven (NL). **VAN ACHT, Victor Martinus Gerardus;** High Tech Campus 5, 5656 AE Eindhoven (NL). **RAMIREZ, Rico Paolo Ochoa;** High Tech Campus 5, 5656 AE Eindhoven (NL). **PANDURANGAN, Palani;** High Tech Campus 5, 5656 AE Eindhoven (NL). **TAN, Boon Teck;** High Tech Campus 5, 5656 AE Eindhoven (NL). **PNG, Luck Wee;** High Tech Campus 5, 5656 AE Eindhoven (NL). **TOH, Joo Pheng;** High Tech Campus 5, 5656 AE Eindhoven (NL). **S S, Prakash;** High Tech Campus 5, 5656 AE Eindhoven (NL). **BOERSMA, Joldert Maria;** High Tech Campus 5, 5656 AE Eindhoven (NL). **VAN AKEN, Timotheus Johannes Maria;** High Tech Campus 5, 5656 AE Eindhoven (NL). **OOSTERHUIS, Joris Pieter;** High Tech Campus 5, 5656 AE Eindhoven (NL). **WISMANS, Antonius Johannes Joseph;** High Tech Campus 5, 5656 AE Eindhoven (NL). **WARMERDAM, Thomas Petrus Hendricus;** High Tech Campus 5, 5656 AE Eindhoven (NL). **RAS, Arnoldus Johannes Martinus Jozeph;** High Tech Campus 5, 5656 AE Eindhoven (NL).

(74) Agent: **PHILIPS INTELLECTUAL PROPERTY & STANDARDS;** High Tech Campus 5, 5656 AE Eindhoven (NL).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ,

(54) Title: PORTABLE TEXTILE TREATMENT DEVICE WITH IMAGE SENSOR AND THERMAL INSULATION MEANS

FIG.11

(57) Abstract: The invention relates to a portable textile treatment device comprising a heatable soleplate (4) intended to be in contact with a textile (TXT) for treating the textile. The heatable soleplate (4) comprises a soleplate opening (H). The device comprises a module (MD) comprising an image sensor (5) for taking an image of the textile to be treated through the soleplate opening (H), and a control unit (8) configured for a) executing an algorithm stored in said portable textile treatment device, using the taken image as an input of the algorithm, to obtain a classification of the textile, and for b) controlling, based on the classification, at least one operating parameter of the portable textile treatment device. The module (MD) and the control unit (8) are integrated within the portable textile treatment device. The image sensor comprises an active surface sensitive to light which is oriented with respect to the surface of the heatable soleplate (4), with an absolute value of an orientation angle being in the range from 15 to 70 degrees. The portable textile treatment device further comprises thermal insulation means arranged in-between the heatable soleplate (4) and the module (MD) for insulating the module (MD) from heat dissipated by the heatable soleplate (4).

[Continued on next page]

WO 2020/254157 A1

Brevet déposé le 17 juin 2019, et publié dans le domaine public depuis décembre 2020 : description très détaillée de l'utilisation d'une caméra associée à un réseau de neurones à convolution dans le fonctionnement d'une centrale vapeur.

La caméra constitue la réelle nouveauté du produit. Un capteur d'image est intégré à l'intérieur du fer à repasser avec une ouverture à travers la semelle pour visualiser le tissu (cf. **encadré n°4**). Le brevet décrit de manière très détaillée plusieurs solutions pour installer cette caméra. Il est intéressant de retenir les caractéristiques suivantes :

- Pour des questions de tenue en température, le capteur n'est pas directement sur la semelle du fer, mais en retrait à l'extrémité d'un tube ; l'angle de vision correspond à une vue de biais de manière à pouvoir capter les différences d'épaisseur dans la trame du textile.
- Le capteur est associé à un système optique (lentille) pour focaliser sur la surface du tissu avec des leds utilisées pour l'éclairage en proche infrarouge de manière à obtenir une image en niveaux de gris.
- La surface visualisée est relativement petite (entre 1x1 mm^2 et 5x5 mm^2), avec une résolution du capteur de la caméra entre 64x64 pixels et 320x320 pixels.

L'exploitation de l'image est réalisée avec un réseau de neurones à convolution. Il permet de calculer 2 informations :

- Le caractère délicat ou non délicat du textile
- Le type de textile : laine, soie, coton, jeans…

Les résultats de ces deux classifications sont exploités pour choisir la température de la semelle, ainsi que la quantité de vapeur. Les règles de pilotage de la température s'appuient sur les caractéristiques de chaque type de textile selon des logiques simples que nous ne détaillerons pas, car elles n'apportent rien pour la compréhension globale du système.

Architecture du réseau à convolution utilisé

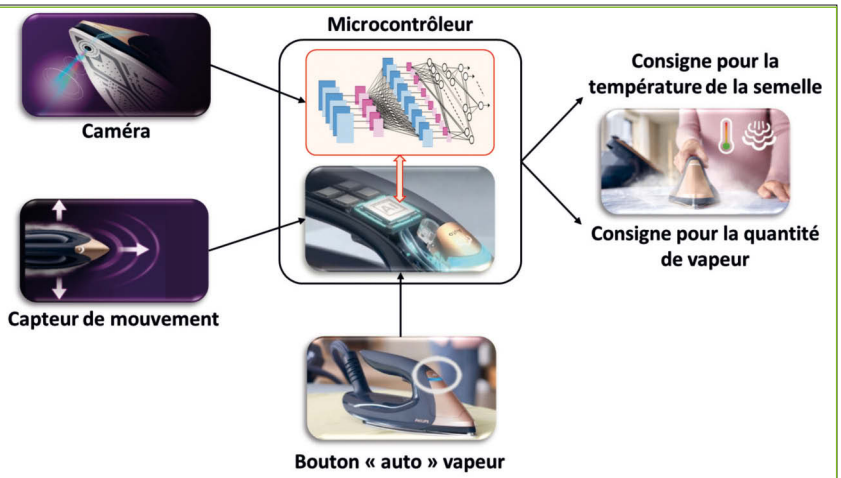
L'utilisation de l'intelligence artificielle correspond à la mise en œuvre du deep learning pour classer des images. Ce choix est pertinent, car les réseaux de neurones sont performants pour ce type de problèmes et nécessitent, dans les phases d'exploitation, des puissances de calculs modestes, comparativement aux autres algorithmes d'interprétation de contenus d'images.

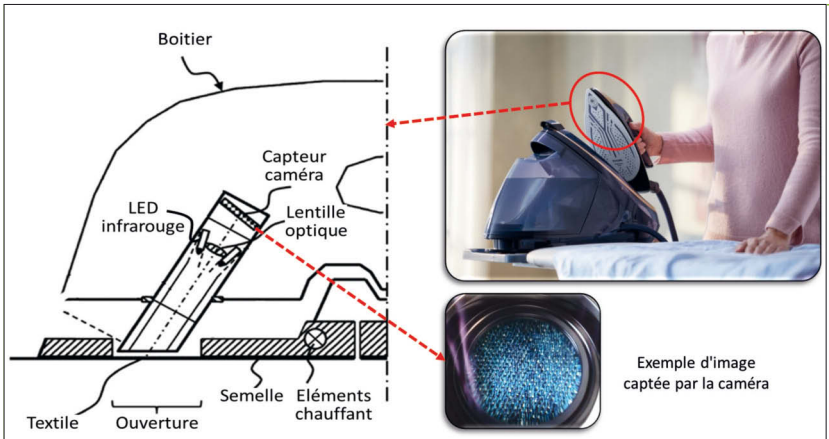
Il est important de bien comprendre la différence entre les phases de mise au point et d'exploitation du réseau. L'exploitation concerne le fonctionnement sur le calculateur cible, ici le microcontrôleur de la centrale vapeur. Ce dernier offre une puissance limitée, mais suffisante pour faire fonc-

Encadré n°3

SYNOPTIQUE DU SYSTÈME EMBARQUÉ DE LA CENTRALE VAPEUR

La centrale vapeur est capable, grâce à des capacités de calcul embarquées, de déterminer la température de la semelle et la quantité de vapeur, à partir de l'image du textile repassé.





Encadré n°4

INTÉGRATION D'UNE CAMÉRA DANS LA CENTRALE VAPEUR

Détails sur l'intégration de la caméra dans la centrale vapeur avec le positionnement du capteur à l'extrémité d'un tube pour le protéger de la température de la semelle et une prise de vue de biais de manière à pouvoir prendre en compte les épaisseurs des fibres du tissu.

tionner en temps réel un réseau. À l'opposé, une capacité de calcul importante est nécessaire lors de la phase de mise au point : des calculateurs spécifiques sont utilisés pour faire fonctionner les algorithmes d'apprentissage, afin de définir les valeurs des milliers de paramètres du réseau.

Le réseau utilisé est détaillé dans l'**encadré n°5**. Il s'agit d'une architecture à convolution similaire à celles décrites dans le n°246. Les images présentées en entrée ont une résolution de 96x96 pixels avec un codage en niveaux de gris. Les étapes de convolution sont constituées de trois phases successives. À chaque étape, les différents traitements sont parallélisés en quatre branches puis les résultats sont fusionnés pour former un nouveau groupe unique d'images. Après chaque phase la taille des images diminue de moitié, mais leur nombre est doublé, ce qui correspond à l'extraction progressive des caractéristiques discriminantes. À la fin de la 3e phase, le résultat est composé de 256 motifs de 6x6 pixels.

Chaque motif est ensuite remplacé par une valeur unique correspondant à la moyenne des valeurs des 36 pixels. La couche de classification exploite les 256 valeurs ainsi obtenues, pour calculer les sorties finales du réseau : la classification délicat/non délicat et la catégorisation du type de textile parmi 32 classes possibles (laine, lin, soie…).

Le brevet ne détaille pas la phase d'apprentissage, mais on peut assez facilement imaginer son déroulement. Dans un premier temps les équipes de recherche ont dû constituer une base d'apprentissage avec des images enregistrées sur un nombre important d'échantillons de textiles composés de qualités de tissage variées et de nombreux motifs et couleurs. Ils n'ont probablement pas uniquement utilisé des tissus neufs, mais aussi des vêtements usagés afin de tenir compte des phénomènes d'usure des fibres.

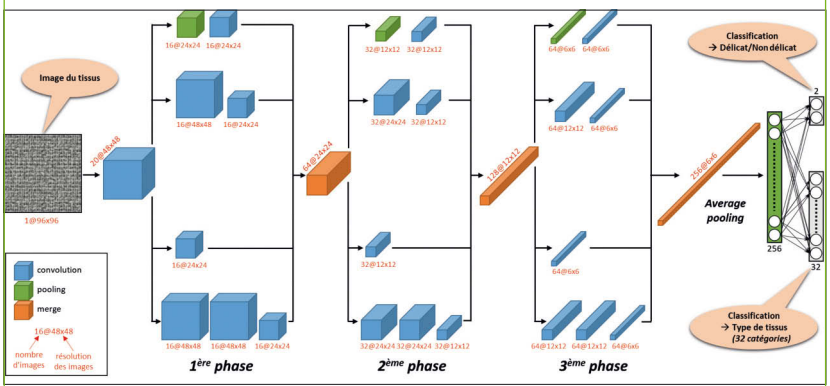
Cette base, sans doute constituée de plusieurs dizaines de milliers d'images, a ensuite été utilisée pour calculer par apprentissage les valeurs des poids et des matrices de convolution du réseau. C'est certainement après plusieurs essais successifs que l'architecture définitive a été retenue et mise en œuvre dans la centrale vapeur.

Conclusion

Cet exemple est intéressant, car il illustre la mise œuvre de la technique du deep learning dans des produits usuels du quotidien. Cela montre que ces solutions d'intelligence artificielle ne sont pas exclusivement réservées aux applications débarquées sur le cloud, ou aux matériels avec de grosses puissances de calculs comme les smartphones haut de gamme ou les véhicules automobiles. Ce type d'application est également possible sur de nombreux équipements « grand public » qui intègrent des microcontrôleurs milieu de gamme. D'ailleurs des fabricants comme ST, NXP ou Intel proposent des outils de développements spécifiques pour les réseaux de neurones [6]. On peut donc supposer que ce type d'applications va se développer, car elles apportent un vrai bénéfice client avec une amélioration des performances des produits.

Encadré n°5

ARCHITECTURE DU RÉSEAU À CONVOLUTION UTILISÉ



Réseau à convolution avec construction de 256 motifs discriminants utilisés pour réaliser 2 classifications séparées : le caractère délicat/non délicat du textile et le type de textile parmi 32 catégories.

Références

DOCUMENTATIONS EN LIGNE :

[1] Descriptif de la centrale vapeur PerfectCare serie 9000 sur le site de la société Philips : https://www.philips.fr/c-p/PSG9030_20/perfectcare-serie-9000
[2] Vidéo de la publicité télévisée : https://www.youtube.com/watch?v=XASmfvgBQ3w
[3] Vidéo du fonctionnement de la centrale vapeur: https://www.youtube.com/watch?v=vITLtP9Yqu4
[4] Site français sur la propriété intellectuelle (INPI) : https://www.inpi.fr/fr
[5] Recherche de brevets sur le site de Google : https://patents.google.com/ (référence WO2020254157A1 pour le brevet lié à la centrale vapeur de Philips)
[6] Outils réseau de neurones pour les microcontrôleurs de la société ST : https://www.st.com/content/st_com/en/ecosystems/stm32-ann.html#stm32sann-overview
[7] École d'ingénierie informatique EPSI (membre de HEP Éducation) : https://www.epsi.fr/

Les testeurs ne sont pas « l'Assurance Qualité » !

Le terme d'ingénieur « QA » pour « Quality Assurance » - traduit Assurance Qualité en français – est de plus en plus fréquent pour nommer le métier de « testeur logiciel ». Cette démocratisation de ce terme tend à indiquer que le testeur est, par sa présence et son action, une « assurance » de la qualité !

Marc Hage Chahine

Key member centre d'expertise de test d'Altran (ITQ) Créateur et animateur du blog La taverne du testeur

Il faut d'abord revenir à ce qu'est une assurance. Je vois personnellement plusieurs définitions qui peuvent convenir :

- Une entité qui est là pour diminuer le plus l'impact d'un incident après qu'il arrive
- Une entité qui valide l'atteinte d'un niveau particulier

Le testeur répond-il à ces définitions ?

Avant de continuer sur cette analyse, il me semble important de noter que ces 2 compréhensions possibles ne le sont qu'en français ! En effet, le terme anglais « Quality Assurance » se traduit plus correctement avec le terme « contrôle qualité ». La mauvaise traduction commune en « Assurance qualité » induit une compréhension différente du titre et donc potentiellement des attentes différentes du métier !

Expérience personnelle sur l'utilisation de mon assurance

Devoir faire fonctionner son assurance est rarement bon signe. Je m'en suis rendu compte quand j'ai récemment dû y faire appel (de nombreuses fois) lors d'un trajet en voiture ! Voici l'histoire de ce trajet :

- Je pars en voiture à 14h pour un trajet de 250km qui prend en moyenne 2h45
- Au bout d'une heure ma voiture perd en puissance et je me retrouve à 50 km/h sur l'autoroute
- Je fais appel à l'assurance et me fais remorquer jusqu'à un garage
- L'assurance me loue une nouvelle voiture que je récupère une voiture chez un loueur connu.
- Je fais les vérifications « usuelles » de la voiture pour regarder s'il y a des problèmes sur les pneus et la carrosserie puis je reprends la route à 18h
- Je m'arrête à 18h45 sur une aire pour manger, il fait encore jour
- Je repars à 19h30 il commence à faire nuit
- J'essaie d'allumer mes feux de croisement, aucun feu (à part les phares, interdits) ne fonctionne
- Je m'arrête de nouveau et fais appel à l'assurance du loueur
- Je suis de nouveau remorqué jusqu'à un garage, mais cette fois-ci il n'y a plus de loueur ouvert (couvre-feu + vendredi soir avec confinement le week-end…)
- L'assurance m'appelle un taxi qui m'amène à destination (arrivée à 22h30)
- Le lendemain, après une matinée au téléphone nous récupérons une nouvelle voiture de location (avec un taxi qui nous amène chez le loueur), mais nous avons cette fois été déclassés

© François Cointe, tiré du livre édité par le CFTL en 2019 "Les tests logiciels en Agile"

- Je vous épargne la suite des péripéties qui n'ont plus vraiment de rapport avec le rétablissement de la situation et qui passent par des incompréhensions, des courses de vélos, des problèmes de charge dans la voiture…

À noter : *je n'ai rien payé à part les réparations sur ma voiture*

Une assurance est là pour diminuer l'impact d'un incident lorsqu'il arrive – ne corresponds pas à l'appellation originelle en anglais

Ce point est primordial. En français, le terme « assurance » n'a pas le même sens qu'en anglais. Dans le français le plus courant, l'assurance est là pour diminuer l'impact d'un incident est le sens premier du mot assurance. Si l'on devait traduire « assurance » en anglais, on se retrouverait avec le terme « insurance ». En français, l'assurance c'est avant tout quelque chose que l'on paye pour nous sortir d'une mauvaise situation imprévue aussi vite et bien que possible en nous permettant de revenir à « la normale ».

On remarque dans mon expérience de trajet en voiture que sur le second point j'ai fait appel à mon assurance, car ma voiture ne fonctionnait plus.

L'assurance a ensuite mis en œuvre différents moyens pour régler mes 2 problèmes en :

- me sortant de l'autoroute où je me trouvais bloqué
- me fournissant une voiture pour pouvoir continuer mon trajet…

En à peine 3 heures (panne vers 15h, nouvelle voiture vers 18h), tout était quasiment revenu à la normale.

De même j'ai de nouveau fait appel à une assurance 1h30 plus tard au point 9. Le problème était similaire (à l'exception que j'étais sorti de l'autoroute pour être au milieu de la campagne), mais la solution à court terme légèrement différente en :

- me sortant de l'endroit où j'étais bloqué
- m'amenant à destination avec un taxi.

Il me restait néanmoins toujours un problème : impossible de faire un trajet retour. J'ai dû attendre le lendemain pour avoir une nouvelle voiture… qui s'est avérée être 1 gamme en dessous de la première et de ma voiture initiale. On a donc ici une dégradation et pas de retour à la normale avant de récupérer ma voiture.

On voit ici ce rôle primordial des assurances. Il y a un problème en production elles sont là pour assurer ce retour à la normale (récupérabilité) aussi vite que possible… et peut passer par des étapes intermédiaires lorsque ce retour à la normale s'avère trop long.

Vous commencez à le ressentir, le testeur logiciel n'a pas du tout ce rôle. Le testeur n'intervient que sur la production et pas pour la réparer. Cette non-adéquation est somme toute logique lorsque l'on se rappelle que le terme initial n'a pas ce sens et que ce dernier, trop souvent interprété comme tel en français, est issu d'une mauvaise traduction de l'anglais au français.

Si l'assurance qualité devait prendre cette définition, alors le rôle d'assurance qualité serait en fait celui de l'Ops de DevOps. Celui qui vérifie les logs, travaille à la détection d'incident et fixe ces incidents, soit de manière temporaire le temps de trouver une solution plus pérenne.

Une assurance permet de valider l'atteinte d'un niveau minimum (de qualité) – correspond à l'appellation originelle en anglais

On est ici sur le point 5 de mon expérience, point qui correspond au sens primaire du terme défini en anglais. Le loueur a des vérifications habituelles qui doivent « assurer » que le véhicule est fonctionnel. De même lors de la récupération du véhicule j'ai également fait des vérifications qui tenaient lieu « d'assurance » tout en considérant que cela devait être minime, car partait du principe que le loueur faisait lui-même des vérifications plus poussées et régulières sur ses véhicules.

Au final, le véhicule qui était assuré d'avoir un niveau de qualité suffisant pour mon trajet s'est avéré avoir un défaut critique (dans mon contexte de nuit)… Il y a clairement eu un « trou dans la raquette »… Et j'espère que la vérification des feux entrera maintenant en compte (suite à une rétrospective ?) lors des vérifications régulières (campagne de régression ?) faites sur les voitures par ce loueur… Dans tous les cas, cette étape est maintenant rentrée dans mes vérifications obligatoires lorsque je loue une voiture.

Comme vous pouvez le remarquer, on est ici totalement dans un rôle qui ressemble au rôle d'un testeur logiciel ! Le testeur logiciel se retrouve à fournir de l'information sur un niveau de qualité suite à diverses vérifications (tests). Néanmoins il est impossible de tout vérifier (exhaustivité impossible des tests) et on ne peut donner qu'un niveau de confiance sur la qualité à partir d'informations incomplètes. Le testeur sera donc amené à faire des erreurs et à devoir améliorer ses processus.

On peut donc remarquer que si l'on prend cette définition d'assurance, le testeur semble parfaitement rentrer dans cette définition. Cela est d'ailleurs totalement cohérent avec le sens du terme en anglais.

Néanmoins, vous aurez également remarqué qu'il n'y a pas qu'un seul intervenant qui a fait des vérifications.

Il y a au moins :

- le garagiste qui fait des vérifications peu fréquentes et est amené à faire des modifications du véhicule qu'on peut assimiler à un développeur. Je suppose que dans mon cas il n'a pas pu détecter le problème, car il n'a pas été amené à faire des vérifications après la panne sur les feux.
- Le « gérant » du parc des voitures à louer qui récupère les voitures et les prête en faisant des vérifications à chaque étape. Cela ressemble pas mal au rôle de testeur.
- Le client (moi) qui a fait quelques vérifications de routine (tests d'acceptation / tests métiers)

Il serait donc, comme on le voit sur l'exemple, de réduire l'assurance qualité au testeur ! L'assurance qualité est l'affaire de beaucoup plus de personnes que simplement le testeur !

Le rôle de testeur se restreint-il à l'assurance de l'atteinte d'un certain niveau de qualité ?

Comme nous venons de le voir, l'assurance qualité ne peut être réduite au rôle/poste de testeur logiciel ! On peut également se poser la question inverse. Le testeur ne fait-il que de l'assurance qualité en fournissant de l'information sur l'évaluation d'un niveau de qualité ?

Non, les testeurs c'est beaucoup plus que cela. Même si l'assurance qualité, dans le sens de l'évaluation de la qualité, est une activité centrale de tout testeur logiciel chaque testeur sera amené à travailler sur d'autres points. Ces points sont très variés et ne sont que très rarement les mêmes d'un testeur à l'autre ou même d'une mission à l'autre, car cela dépend fortement du contexte.

Il est par exemple possible de penser à :

- Travailler à la prévention des défauts : même si cela se rapproche de l'assurance qualité on est ici sur une phase en amont
- Travailler à l'amélioration des processus de l'assurance qualité comme l'automatisation de certaines activités
- Suivre la production et bien connaître les utilisateurs
- Faire des propositions d'amélioration du logiciel testé
- Diffuser l'esprit de la qualité aux différents acteurs du projet
- Contribuer, pour le testeur Agile et lorsque nécessaire, à toute activité utile à l'équipe
- Documente le logiciel d'un point de vue technique (spécifications) ou utilisateurs
- Analyse, crée des liens dans le but d'assurer une amélioration continue…

Conclusion

Le sens des mots est particulièrement important, car l'emploi de certains mots fait passer certaines idées. Cela est évidem-

ment le cas lors des traductions faites trop vite. Le terme « QA », traduit « assurance qualité » en français, est là pour nous le rappeler.
Si l'on devait partir de la définition première de l'assurance en français, la personne en charge de l'assurance qualité serait l'Ops ! L'Ops est le garant de la production et assure une remise en état aussi rapide que possible du logiciel lors de l'apparition d'un incident… Cette mécompréhension encourage d'ailleurs de fausses idées comme celle laissant penser qu'après les tests il n'y a pas de bugs.
Fort heureusement, il n'y a pas qu'un sens au terme « assurance » en français. Assurance peut être compris comme de l'information. Dans ce cas il est vrai que le testeur semble parfaitement correspondre à cette définition… Cela tombe bien, car c'est le sens qu'il a en anglais ! Cette possibilité de traduction du mot assurance a d'ailleurs peut être poussé à conserver le terme « assurance qualité » en français plutôt qu'une traduction plus proche : « contrôle qualité ». On peut également penser que le fait que le terme « contrôle » ait une connotation plutôt négative ait joué.

En creusant un peu sur cette bonne définition du terme il devient cependant évident que le testeur n'est pas le seul à contribuer à cette assurance, mais aussi que le testeur est amené à faire plus que cette assurance… Le rôle de QA, dans le sens originel (anglais) de donner de l'information, reste cependant le rôle central de tout testeur. Le terme ingénieur QA est donc un terme que je trouve légitime à condition de se rappeler son origine anglaise. De plus il ne faut pas non plus tomber dans les travers que l'on a eu avec le terme testeur (le testeur ne fait pas tous les tests et ne passe pas son temps à exécuter des tests !) et ne pas oublier que tout le monde contribue à l'assurance qualité, mais aussi le fait que l'ingénieur QA ne fait pas que du contrôle qualité.

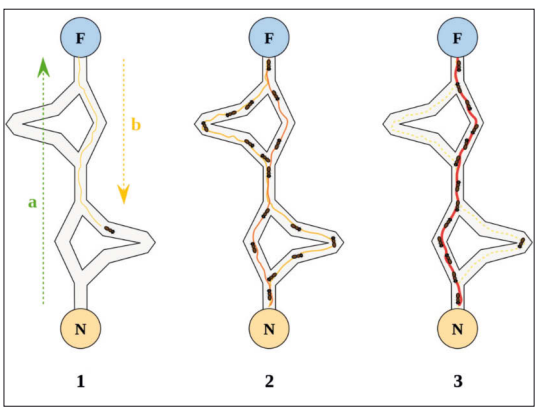
Un grand merci à Olivier Denoo, Benjamin Butel et Christophe Moustier pour leurs retours et partages. Sans eux l'article serait beaucoup plus pauvre et n'aurait pas d'analyse sur l'impact des mots et n'aurait pas proposé d'analyse sur le terme original anglais !

Intelligence en essaim et industrie logicielle

Nous nous enorgueillissons souvent, en tant qu'être humain, d'être particulièrement intelligent et capables de résoudre des problèmes complexes ! Cet orgueil est-il vraiment bien placé ? L'étude de comportement d'animaux « a priori » très peu intelligents nous montre que ces derniers arrivent néanmoins, de façon récurrente, à résoudre des problèmes pourtant très complexes !

Parmi ces « exploits » il y a notamment les fourmis qui sont capables très rapidement de toujours prendre le chemin le plus court entre leur fourmilière et la source de nourriture : **figure 1**.
De même, il y a les oiseaux et les poissons qui arrivent à voler ou nager de manière très concentrée sans pour autant entrer en collision.
Ces prouesses ne sont pas le fruit du hasard, mais d'une « intelligence en essaim ». Les animaux ne voient pas le problème dans sa globalité, mais obéissent à des règles simples tout en reposant sur un moyen de communication efficace.

Figure 1
Source: Wikipédia



Pour les fourmis c'est suivre la piste avec la plus forte concentration de phéromones, pour les oiseaux c'est rester à une distance minimale des ses plus proches voisins !
On peut également penser aux « hola », aux retournements de pancartes dans les événements sportifs et aux chorégraphies complexes si on souhaite analyser des comportements humains. De simples règles permettent donc d'accomplir des exploits. C'est en partant de cette observation, et plus précisément de l'exemple des fourmis, que « la robotique en essaim » est née.

La robotique en essaim

La robotique en essaim tente de proposer des solutions à des problèmes complexes, non pas en utilisant une intelligence artificielle complexe, mais en impliquant de nombreux robots dont l'intelligence est limitée (programmes avec des règles simples).
La robotique en essaim a vu le jour en observant les fourmis. La résolution de la recherche du chemin le plus court a réussi à être résolue avec des robots en utilisant des traces lumineuses pour remplacer les phéromones : **figures 2 et 3**.
Les recherches actuelles se concentrent plus sur les vols des oiseaux afin de faire voler des essaims de drones en évitant les accidents et en proposant une formation définie (il existe de nombreuses vidéos depuis 2018 à ce sujet). Les applica-

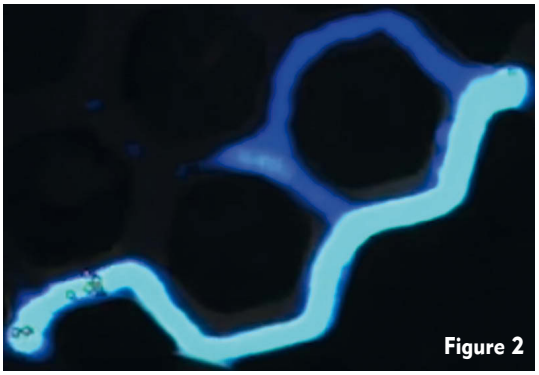


Figure 2

Source: vidéo fouloscopie *Source: vidéo fouloscopie*

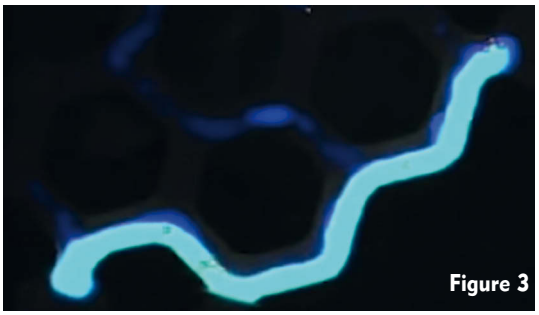


Figure 3

tions envisagées se font au niveau de l'exploration avec de la recherche de victimes après une catastrophe, la recherche de fruits malades pour une agriculture plus intelligente ou encore la détection de poussière pour nettoyer les appartements.

Les perspectives envisagées ne se limitent pas à de l'exploration, mais aussi à la médecine avec, à l'horizon 2040 (cela reste une estimation), des robots en essaim utilisés pour détruire les cellules cancéreuses.

Bref, la robotique en essaim semble promise à un grand avenir, mais, concrètement, à quoi cela peut ressembler actuellement ? Un élément de réponse avec l'expérience menée par Mehdi Moussaid : https://www.youtube.com/channel/UCLXD-NUO03EQ80VmD9nQBHPg

Expérience de robotique en essaim

L'idée de l'expérience est de proposer un tournoi utilisant des robots ayant tous le même programme et devant s'acquitter d'une tâche le plus rapidement possible.

Les 12 robots doivent trouver une cible dans l'arène et « l'éliminer ». Cette tâche se déroule donc en 2 étapes :

- **L'exploration** pour la découvrir la cible (vision et communication restreintes de chaque robot) ;
- **La destruction** où 2 actions sont possibles : attaquer ou prévenir les autres robots. La destruction est plus rapide si plus de robots attaquent.

Ce défi se fait avec les contraintes suivantes :

- Pour trouver la cible, un robot doit rouler dessus
- L'information sur la localisation peut être fournie uniquement par un robot qui connaît cette dernière
- Le rayon de communication des robots est limité

Enfin, « l'arène » de recherche et la position de départ du robot sont toujours les mêmes. Voici un schéma permettant de résumer le principe de cette expérience : **Figure 4**

Le fonctionnement du tournoi est le même que celui des tournois de tennis en prenant le meilleur des 10 manches (11 si égalité) entre 2 algorithmes.

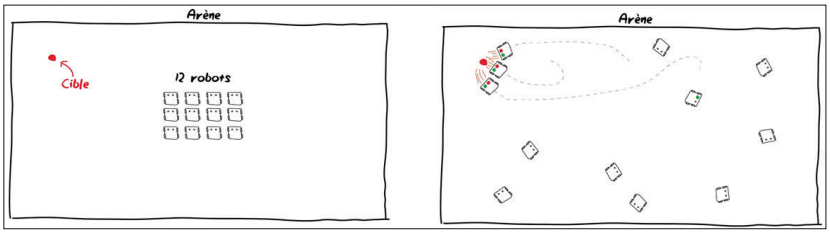


Figure 4 *Source: vidéo fouloscopie*

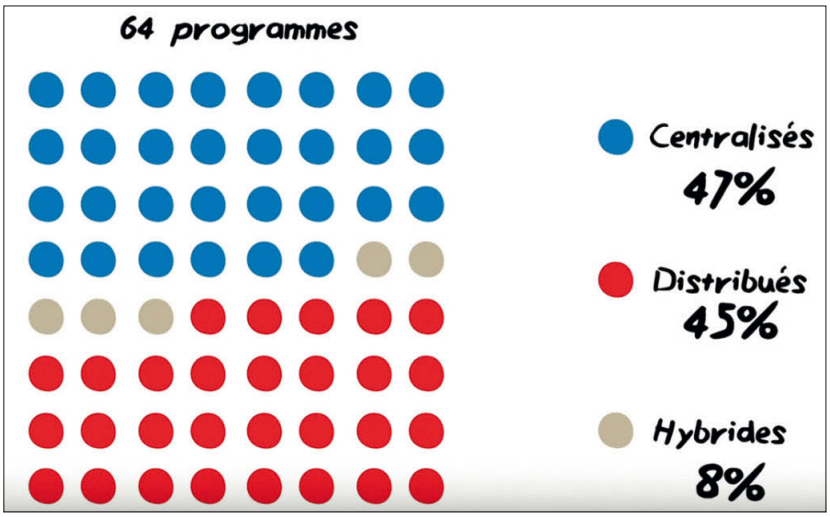


Figure 5 *Source: vidéo fouloscopie*

À noter : *cette expérience est possible grâce au « robotarium » qui permet de programmer des robots à distance et de voir leur comportement. Le robotarium a été créé par Magnus Egerstedt. Il propose aux chercheurs du monde entier de prendre le contrôle de robots à distance pour conduire des expériences scientifiques. Le robotarium est accessible à des visiteurs qui souhaitent voir en direct le comportement des robots.*

Algorithmes proposés

De nombreuses personnes ont participé à l'expérience. Au total, 64 algorithmes ont été proposés. Ces algorithmes peuvent être catégorisés en 3 familles (dont 2 principales représentant chacune environ 45 %) :

Centralisés

- Approche militaire : on décide à l'avance comment vont bouger les robots
- Très peu de marge d'improvisation, le programmeur est un chef d'orchestre

Distribués

- Directives simples et assez vagues (ex : rester à une certaine distance + déplacement libre)
- Pas de contrôle sur l'emplacement exact

« Hybrides »

- Globalement un mixte des 2 approches avec des résultats globaux qui sont en moyenne entre les 2 types d'algorithmes principaux

Cela donne cette répartition : **Figure 5**

Résultats de l'expérience et analyse

Le résultat de ce tournoi donne une victoire avec une courte avance (6 à 5) pour un algorithme distribué. Sur le meilleur algorithme centralisé. Néanmoins la 2e et la 3e place sont occupées par des algorithmes centralisés et la moyenne de

destruction des cibles est largement en faveur des algorithmes centralisés qui mettent en moyenne 30 secondes de moins que les algorithmes distribués pour détruire la cible : **Figure 6**

Ces résultats montrent qu'avec un contexte figé et connu il est généralement plus intéressant d'adopter une approche militaire, très cadrée.

Si l'on pousse l'expérience plus loin et que l'on décide de modifier 1 paramètre, la position de départ comme ci-dessous : **figure 7**

On remarque alors que la performance des algorithmes centralisés est très fortement dégradée contrairement à celle des algorithmes distribués : **figure 8**

Ce résultat semble logique, car comme le dit Mehdi Moussaid dans sa vidéo : « *Quand on décentralise la prise de décision, le groupe peut s'adapter beaucoup plus facilement à un nouvel environnement parce que le chef n'a pas besoin d'intervenir à chaque fois pour repenser la stratégie et c'est probablement pour ça que ça marche aussi bien dans la nature* »

Au-delà de cette « logique », il me semble intéressant de faire un parallèle avec les métiers de l'IT.

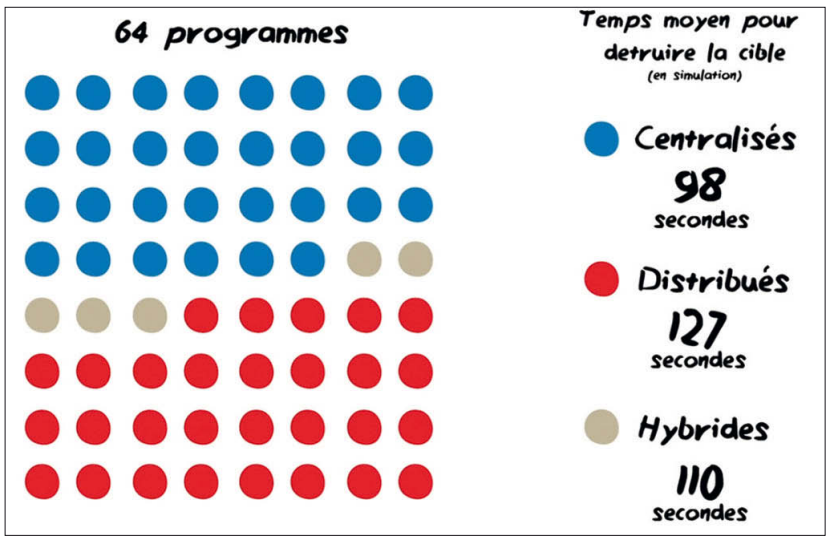


Figure 6 *Source: vidéo fouloscopie*

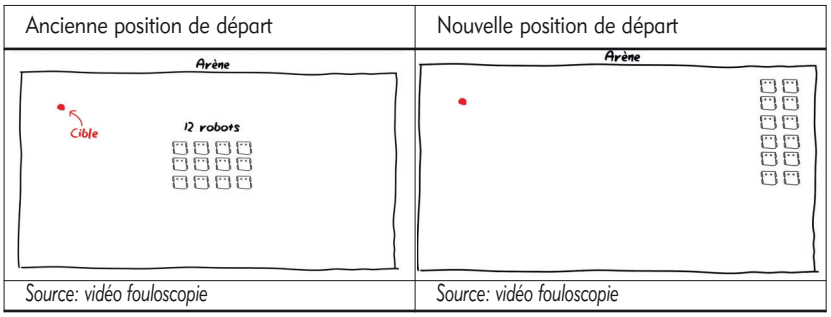


Figure 7

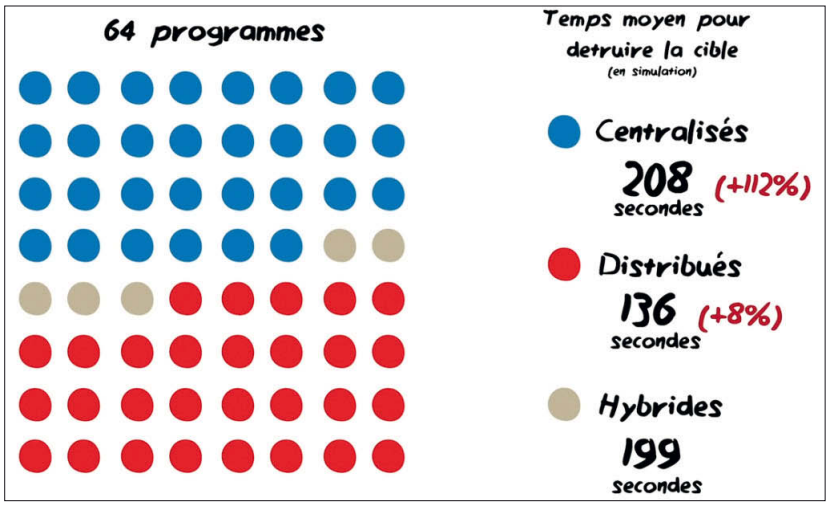


Figure 8 *Source: fouloscopie*

Cycle en V vs Agile

Le duel « Cycle en V vs Agile » est un duel qui est actuellement largement à l'avantage de l'Agile. Ce n'était pas le cas il y a encore 10 ans ! Au-delà de la théorie et du discours pro Agile (que je tiens également) sur la capacité à répondre au changement, s'adapter et livrer de la valeur plus vite, il me semble important d'analyser des raisons possibles de cette évolution grâce aux résultats présentés précédemment.

Pour rappel, dans l'expérience avec les robots, le modèle distribué avec peu de planification, des règles simples et une relative liberté des robots s'adapte en moyenne mieux au changement que le modèle centralisé (« militaire »).

Ce changement est tangible dans notre vie de tous les jours avec une évolution très rapide des demandes des utilisateurs, des technologies ou même des terminaux utilisant les logiciels c'est d'ailleurs un point très important remonté dans le framework SAFe (https://www.scaledagileframework.com/). Ce contexte semble favoriser les algorithmes dits « distribués » qui sont peu impactés par ces évolutions du contexte. **Figure 9**

Si on analyse brièvement les caractéristiques du cycle en V et de l'Agile on peut arriver à ce tableau :

	Cycle en V	Agile
Caractéristiques	Planification / prévision Chacun à un rôle défini	Recherche de retour d'expérience Rôle définis en fonction des besoins
Zone de performance	Contexte figé (besoin, périmètre, technologie…)	Besoin de prise de connaissance, environnement / contexte changeant
Zone moins performante	Contexte évolutif	Contexte figé

L'Agile semble donc être relativement proche d'un modèle distribué et le cycle en V d'un modèle centralisé.

Connaissant cela et le contexte actuel, il semble logique que l'Agile, plus performant dans notre monde moderne soit privilégié ! Le cycle en V reste néanmoins performant dans des situations maintenant minoritaires où l'on maîtrise l'environnement et que l'on n'a pas besoin de s'adapter à des changements fréquents.

Agile à l'échelle

La liberté est très importante afin de s'adapter. Néanmoins, la performance des algorithmes distribués avec cette liberté n'existe que s'il y a une collaboration de tous les instants avec des règles communes même si elles restent simples !

Pour rappel, les robots en essaim avec un algorithme distribué proposent ces caractéristiques :

• Décentralisation partielle de la prise de décision
• Tous les robots ont le même programme « simple »
  - Simple, mais permettant la collaboration
• Chaque élément ne voit pas le problème dans sa globalité

Il est alors très aisé de faire un rapprochement avec l'Agile à l'échelle (ou même dans une moindre mesure à des équipes agiles) : **figure 10**

Il est bon de noter que dans ce cas il faut savoir trouver le juste milieu entre des règles communes et l'indépendance de l'équipe, les 2 extrêmes étant néfastes (pas de règles = perte de temps et manque de synchronisation ; trop de règles = retomber dans les travers du « cycle en V »). *Pour aller plus loin sur le sujet vous pouvez vous pencher sur le concept et Panarchie et de double-boucle d'apprentissage présenté dans le livre de Christophe Moustier.*

Il n'est d'ailleurs pas étonnant que des modèles comme SAFe et Spotify accordent une part importante à ce socle commun ou lieu de partage. SAFe avec le « system team » et les communautés de pratiques, Spotify avec les guildes et les tribus.

Les tests

Le parallèle peut également être fait avec le test : **figure 11**

Les robots en essaim, tout comme les tests, doivent **s'adapter au contexte** (dans l'expérience présentée, cela peut être le nombre de robots, la taille et la couleur de la cible, la disposition initiale ou encore la taille et forme de l'arène)Pour les robots en essaim, tout comme pour les tests, **la qualité du résultat final dépend de tous**.Pour les robots en essaim, tout comme pour les tests, personne ne peut tout faire ou tout savoir. **Chacun fait sa part, communique et a sa vision, son point de vue**.Le test a d'ailleurs déjà développé des techniques permettant de répondre à ces difficultés. On peut par exemple citer :

• Les tests exploratoires

Afin de s'adapter au contexte et comme l'a montré l'expérience, il faut avoir une décision au moins en partie décentralisée. Il ne faut pas tenter de tout prévoir en amont. Les tests exploratoires offrent une solution quasi parfaite à cette problématique en offrant une totale liberté au testeur dans un certain cadre défini par la charte de tests exploratoires (temps dédié, périmètre de test, persona…). Les tests exploratoires sont un parfait complément aux campagnes plus classiques en ce qu'ils offrent une grande flexibilité au testeur qui va pouvoir multiplier les tests et les adapter au fil de l'eau. Finalement, les tests exploratoires offrent un gain de temps très important tout en renforçant la qualité globale du logiciel… Et en luttant contre le paradoxe du pesticide.

Il n'est pas étonnant que ces tests soient considérés comme essentiels pour beaucoup de professionnels du test dont Frédéric Assante Di Capillo qui en a fait un article sur le blog de la taverne du testeur : https://latavernedutesteur.fr/2021/02/25/les-tests-exploratoires-une-obligation-dans-lagilite/

Il me semble également bon de rappeler que les tests exploratoires dépendent fortement du testeur (ce dernier doit de préférence bien connaître le produit et être un minimum expérimenté) et qu'ils sont généralement moins adaptés pour les logiciels « critiques » ou fortement contraints (contexte légal ou autre). Dans ce cas-là on peut penser à retourner sur une solution "centralisée" qui est meilleure "en moyenne" dans un contexte spécifique, connu et stable.

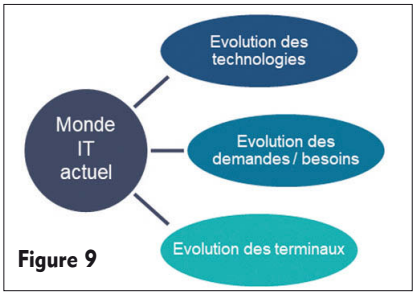


Figure 9

Figure 10

L'Agile à l'échelle peut s'inspirer de bonnes pratiques des essaims:
• Les équipes doivent garder de l'indépendance, mais aussi adopter un socle commun (communautés de pratiques, règles simples, outils…)
• Les équipes sont indépendantes, mais doivent communiquer (synchronisation possible avec une équipe transverse)

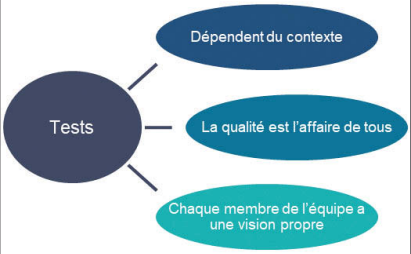


Figure 11

Les robots en essaim, tout comme les tests, doivent s'adapter au contexte (dans l'expérience présentée, cela peut être le nombre de robots, la taille et la couleur de la cible, la disposition initiale ou encore la taille et forme de l'arène)
Pour les robots en essaim, tout comme pour les tests, la qualité du résultat final dépend de tous.
Pour les robots en essaim, tout comme pour les tests, personne ne peut tout faire ou tout

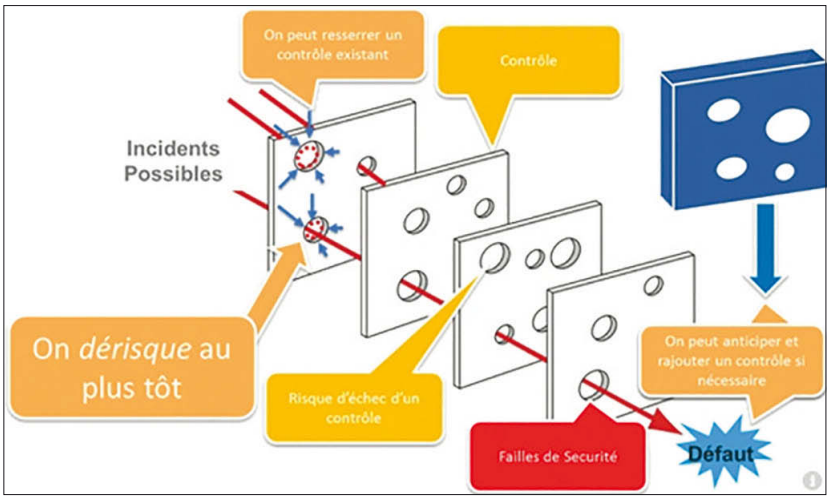


Figure 12 *Source: wiki fan de test*

• Le modèle en tranches d'emmental **Figure 12**

Le modèle en tranches d'emmental est plus une représentation qu'une vraie technique. Ce modèle est également applicable aux "niveaux de test" et ainsi expliquer leur raison d'être. Il permet néanmoins de montrer simplement qu'il n'existe pas de campagne unique permettant de détecter l'ensemble des anomalies. Une détection efficace des anomalies ne peut se faire qu'en multipliant les types de tests à différents niveaux. Chaque acteur du projet doit faire des tests à son niveau et œuvrer pour la qualité du logiciel. Sans

ce travail de tous les instants et effectué par tous, la perte de qualité s'en fait grandement ressentir.

Cette complémentarité des tests et des acteurs est d'ailleurs régulièrement mise en avant dans les articles ayant pour sujet la qualité. Je pense notamment à l'article de Benjamin Butel « Développeurs et testeurs : équipe de choc (https://lataverne-du-testeur.fr/2020/01/14/developpeurs-et-testeurs-equipe-de-choc/) où une partie de la conclusion est :

« *Développeurs + testeurs + bonne communication = produit qualité* »

Vous noterez que la communication reste prépondérante… tout comme pour les robots en essaim.

• Le crowdtesting

Le crowdtesting est une pratique de test relativement récente (une dizaine d'années). Le principe, tout comme celui d'un bêta test, est de faire intervenir des utilisateurs potentiels pour tester un logiciel (ou une version), en général avant sa mise effective sur le marché. Là où le crowdtesting diffère d'un bêta, c'est que les crowdtesteurs sont rémunérés et cadrés dans leur démarche de test ce qui est assez rare dans les bêta. Leur but n'est pas forcément de simplement donner un avis ou juste utiliser le logiciel, mais bien de le tester. On peut d'ailleurs pousser le parallèle avec la robotique en essaim en comparant chaque crowdtesteur à un robot de l'essaim. L'essaim de testeurs est « programmé » de manière assez faible, ce qui lui permet d'acquérir un certain niveau de liberté.

D'un point de vue technique, une mise en place d'une campagne de crowdtesting peut ressembler à cela : **Figure 13**

Ces campagnes remontent des anomalies de différents types, mais surtout des anomalies non trouvées par les équipes de développement (au sens large) et connaissant parfaitement leur logiciel.

Si vous voulez en savoir plus sur le Crowdtesting, je vous invite à regarder le Webinaire « L'expérience du Crowdtesting (https://www.youtube.com/watch?v=Fd8xt75J4nE&) disponible sur la chaîne YouTube du blog de la taverne du testeur.

Pour résumer, le crowdtesting est peut se rapprocher du « test en essaim » en prenant en compte ces caractéristiques :

Robots en essaim	Crowdtesting
• Algorithme dispersé qui permet de s'adapter au contexte • Tous les robots ont le même programme « simple » • Simple, mais qui permet ensemble de répondre à un problème complexe • Chaque élément ne voit pas le problème dans sa globalité	• Propose des tests non prévus, mais orientés et exécutés par des testeurs indépendants • Complémentaires aux tests scriptés • Tests scriptés « simples » • Tests exploratoires en plus

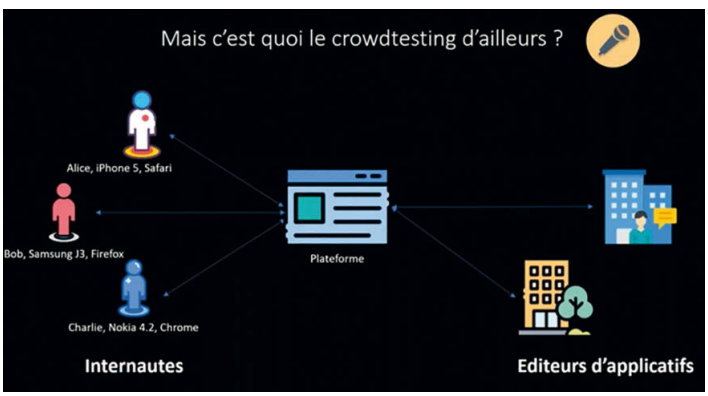


Figure 13
Source: webinaire de la taverne du testeur « L'expérience du Crowdtesting »

Conclusion

La phrase de Mehdi Moussaid (chercheur dans l'étude du comportement des foules (appelé fouloscopie) et créateur/animateur de la chaîne éponyme) :

« *Quand on décentralise la prise de décision, le groupe peut s'adapter beaucoup plus facilement à un nouvel environnement parce que le chef n'a pas besoin d'intervenir à chaque fois pour repenser la stratégie et c'est probablement pour ça que ça marche aussi bien dans la nature* »

me semble un très bon point de départ pour penser en termes d'essaim et plus généralement en termes d'efficacité dans de nombreux domaines ce qui inclut évidemment le logiciel. Dans cet article je ne propose que des pistes de réflexion à la suite des résultats d'une expérience. Il va sans dire que nous pouvons pousser la réflexion plus loin sur les domaines de l'Agile, de l'Agile à l'échelle, du test ou tout autre sujet de notre vie quotidienne.

S'il y a un point qui ressort selon moi de cette analyse de l'intelligence en essaim, c'est bien qu'il n'y a pas forcément besoin d'être un génie pour accomplir de grandes choses ! Il n'y a pas forcément besoin non plus d'avoir quelqu'un capable de tout prévoir (de toute manière, cela n'existe pas). Non, pour accomplir des exploits, il faut surtout un travail collaboratif, de la communication qui soit coordonnée par des règles communes simples et un objectif commun. Il est bon de noter que cela ne s'applique pas uniquement à l'homme, mais aussi aux animaux ou encore les neurones d'un cerveau !

Au final, comme vous l'avez remarqué, par intelligence en essaim on peut également comprendre intelligence collective… On voit alors l'intérêt de travailler ensemble, de former une équipe afin d'arriver à atteindre nos objectifs. Cela transparaît particulièrement bien à travers ces phrases de Michael Jordan :

"*There are plenty of teams in every sport that have great players and never win titles. Most of the time, those players aren't willing to sacrifice for the greater good of the team. The funny thing is, in the end, their unwillingness to sacrifice only makes individual goals more difficult to achieve.*"

que l'on pourrait traduire par :

"Il y a beaucoup d'équipes dans tous les sports qui ont de grands joueurs et qui ne gagnent jamais de titres. La plupart du temps, ces joueurs ne sont pas prêts à se sacrifier pour le plus grand bien de l'équipe. Le plus drôle, c'est qu'au bout du compte, leur refus de se sacrifier ne fait que rendre les objectifs individuels plus difficiles à atteindre."

Un grand merci à Zoé Thivet, Benjamin Butel, Olivier Denoo, Christophe Moustier et Michael Granier pour leurs relectures et propositions d'améliorations pour cet article qui… est aussi issu d'un travail collaboratif !

Base de données convergée Oracle : Relationnel et JSON, mais aussi Graph, Spatial, XML, Texte… PARTIE 2

Après plus de 4 décennies d'existence, la base de données Oracle continue de se renouveler. Depuis plus de 7 ans, le support des données JSON n'a fait que s'enrichir. Avec la version 19c et sa mise à disposition via des services managés, notamment Autonomous JSON Database, les développeurs ont désormais accès à la base de données convergée par excellence. Elle sait traiter de nombreux types de données (JSON, relationnelles, graph, spatial, XML, texte…) pour des charges de travail multiples (OLTP, OLAP, IoT, blockchain, streaming d'événements, machine learning…), avec des APIs simples (SODA, REST, SQL, PGQL…) et ce pour tous les langages modernes du marché. Et 2021 vient avec son lot de nouveautés…

Loïc Lefèvre

Développeur depuis l'âge de 8 ans, je multiplie mes expériences professionnelles depuis plus de 20 ans : startups, grands comptes de la finance et actuellement chez Oracle. Fullstack developer, architecte cloud et DBA applicatif, j'adore les challenges liés au développement d'applications modernes et à la gestion de l'information. Désormais Product Manager pour la base de données Oracle, je participe à l'élaboration des nouvelles fonctionnalités pour tous les développeurs : frontend, backend, low-code, data engineer, business analyst et même data scientist !

@Loic_Lefevre

Nouveau type de données JSON

L'une des dernières nouveautés de la base de données Oracle est le nouveau type de données JSON. Supporté à partir de la version 21c (à partir de la version 19c avec les bases de données autonomes), il utilise un nouveau type d'encodage : OSON. Ce type d'encodage se veut une amélioration du format binaire BSON rendu populaire par MongoDB lui-même plus performant que JSONB utilisé par PostgreSQL. Basé sur une représentation arborescente, vous pouvez voir un exemple de document JSON simple encodé dans un tableau d'octets au format OSON. On peut y voir 3 pointeurs permettant de naviguer par sauts dans la hiérarchie. **Figure 19**

Bénéfice #1 : performance de requêtage

L'encodage OSON, structure arborescente, supporte la navigation par sauts. Ainsi, si l'on désire récupérer uniquement la valeur du champ "name" du deuxième objet JSON du tableau "items" (évaluation du chemin JSON : items[1].name) alors la base de données va sauter directement à la valeur "PC" sans parser et analyser la représentation texte du document JSON. Pour de petits documents JSON, le gain peut sembler modeste, mais pour des documents JSON typiques pouvant avoir des centaines de champs et tableaux d'éléments imbriqués, éviter l'analyse de champs non recherchés est beaucoup plus efficace que de parser *systématiquement* du texte.

Bénéfice #2 : mises à jour plus efficaces

Dans beaucoup d'applications, on peut souhaiter ne mettre à jour que certains des champs d'un document JSON ; on parle alors de mise à jour partielle. Par exemple, vous pourriez vouloir mettre à jour le champ "id" de la valeur "CDEG4" à "CDEG52" dans l'exemple. La base de données réalise des mises à jour partielles du format OSON dès qu'elle le peut, en ne modifiant qu'un sous-ensemble des nœuds de la structure JSON, même si les nouvelles données occupent plus d'espace. En effet, même pour des opérations de mise à jour complexes, le remplacement de tout le document n'est pas nécessaire. Les mises à jour partielles du format OSON améliorent ainsi encore les performances, et ce proportionnellement à la taille des documents. La réduction des I/O notamment concernant les journaux de transaction rend l'application beaucoup plus scalable.

Bénéfice #3 : réduction de l'espace de stockage

Souvent, les documents JSON de taille moyenne à grande (plusieurs Ko à plusieurs Mo) contiennent des objets imbriqués ainsi que des tableaux d'objets. Par exemple, le champ "name" peut être répété de nombreuses fois. Le format OSON, se sert d'un dictionnaire pour ne stocker qu'une seule fois le nom du champ et ainsi réduire la taille des documents. D'autre part, des algorithmes de compression peuvent également être mis en œuvre. **Figure 20**

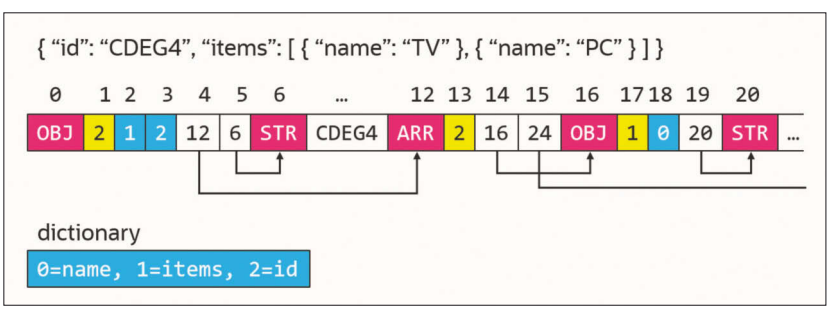
Bénéfice #4 : encodage natif des valeurs

Les valeurs des champs de vos documents JSON utilisent l'encodage natif de la base de données Oracle : number, date, timestamp, binary float/double, year-month, day-time intervals, binary. Par conséquent, il n'y a pas de conversion de données lors de l'extraction des valeurs (lors du requêtage notamment).

Bénéfice #5 : format identique entre le client et la base de données

Le format OSON est supporté dans la base de données et les drivers SODA. L'encodage et le décodage peuvent être réalisés au niveau du client. Ceci permet notamment de réduire

Figure 19



la consommation de ressources CPU côté serveur de base de données. De plus, ces drivers savent également utiliser cette navigation par saut.

Bénéfice #6 : intégration avec toutes les autres fonctionnalités et les autres types de données

C'est sans doute le bénéfice le plus important qu'apporte ce nouveau type de données JSON avec ce nouvel encodage OSON, une intégration avec toutes les autres fonctionnalités de la base de données : index basés sur des fonctions, index pour recherches full-text, index sur GeoJSON pour recherches spatiales, vues matérialisées avec JSON_TABLE(), Data-Guide JSON pour analyser la structure (le schéma) d'une collection, requêtes SQL parallèles, accélérateur mémoire en mode colonne des données JSON pour faire de l'analytique temps réel, traitement JSON au niveau du stockage des infrastructures Exadata… Toutes les fonctionnalités ont été mises à jour pour tirer profit du format OSON pour accélérer le traitement des données JSON et relationnelles… et mêmes pour les combiner entre elles ! Pour plus d'informations, vous pouvez consulter en ligne la session de la VLDB 2020 : *Native JSON Datatype Support: Maturing SQL and NoSQL convergence in Oracle Database*.

Figure 20

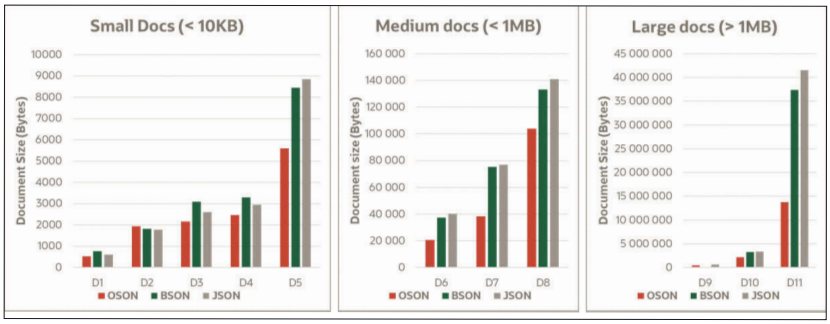


Figure 21

Capability	JSON	Relational
Aggregation + Analytic Functions	✓	✓
Materialized Views	✓	✓
Triggers	✓	✓
Scalable Joins	✓	✓
ACID transactions	✓	✓
Partitioning	✓	✓
In-Memory Columnar Processing	✓	✓
Machine Learning	✓	✓
Advanced Security	✓	✓
Real-Application Cluster	✓	✓
Sharding	✓	✓
Data Guard	✓	✓
GoldenGate replication	✓	✓

Figure 22

SQL Operators	JSON Functions	Returns
SELECT	JSON_QUERY()	*Fragments* of a JSON document
	JSON_VALUE()	Scalar value
	CASE JSON_EXISTS()	False if no JSON value is matched
FROM	JSON_TABLE() (row source)	Specific JSON data projected to relational columns and new rows in case of JSON array
WHERE	JSON_EXISTS()	False if no JSON values are matched
	JSON_QUERY()	*Fragments* of a JSON document
	JSON_VALUE()	Scalar value
	JSON_TEXTCONTAINS()	Row(s) matching the Full-Text search expr.
GROUP BY	JSON_VALUE()	*Scalar value*
ORDER BY	JSON_VALUE()	*Scalar value*

Combiner vos données JSON et relationnelles

C'est LE point fort d'une base de données convergée : simplifier la gestion des données quelles qu'elles soient. Ainsi, la capacité à pouvoir gérer via votre API préférée un modèle de données tout en pouvant utiliser le langage SQL pour combiner, mélanger, mixer différents modèles de données entre eux est très puissant et les impacts sont véritablement là :

- Moins de duplication de données et donc des économies au niveau du stockage,
- Meilleure sécurité : standardisation des politiques, pas de maillon technologique plus faible qu'un autre, surface d'attaque limitée (par l'absence de duplication et de déplacement des données),
- Simplification des processus d'intégration de données JSON et relationnelles : que l'on pourrait résumer à une jointure en SQL (et non une jointure, ce n'est pas lent),
- Simplification des opérations : une technologie maîtrisée avec des process d'exploitation connus et unifiés,
- Intégration des fonctionnalités communes aux différents types de données **Figure 21**

Opérateurs JSON en SQL

Pour appréhender les données de type JSON en SQL, rien ne vaut la "dot notation" :

```
-- Type natif JSON à partir de la version 21c
create table emp ( data JSON );
-- Type alternatif BLOB avec contrainte IS JSON à partir de la version 12c
--
-- create table emp (
--   data BLOB check (data IS JSON)
-- )
-- lob(data) store as (cache);
insert into emp (data) values
( '{"name": "Loïc",
  "job": "PM",
  "salary": 20000,
  "created": "2021-04-17T10:05:00.000Z",
  "email": "loic@oracle.com",
  "phones": [ {"type": "home", "num": "+33123456789"},
              {"type": "mobile", "num": "+33678901234"}
            ]
 }' );

-- Projection des valeurs JSON en colonnes relationnelles : "dot notation"
select e.data.name.string(),
    e.data.job.string(),
    e.data.salary.number(),
    e.data.created.timestamp(),
    e.data.email.string()
 from emp e;
```

Plusieurs opérateurs JSON sont disponibles pour être utilisés dans vos requêtes SQL : **Figure 22**

L'un des plus intéressants est sans doute JSON_TABLE() per-

mettant de "dérouler" les valeurs imbriquées dans un tableau d'objets, par exemple pour le champ "phones" :

```
-- Projection des valeurs JSON en colonnes relationnelles avec valeurs
 imbriquées
-- Notation avec opérateur standard JSON_TABLE()
select e.*
 from emp,
    JSON_TABLE( emp.data, '$' columns (
            name,
            job,
            salary,
            nested phones[*] columns (
              type,
              num
            )
           )
          ) e;
```

Figure 23

```
-- Projection des valeurs JSON en colonnes relationnelles avec valeurs
 imbriquées
-- Notation simplifiée (21c+)
select e.*
 from emp nested data columns (
        name,
        job,
        salary,
        nested phones[*] columns (
          type,
          num
        )
    ) e;
```

Générer du JSON depuis des données relationnelles

Il existe également des opérateurs pour créer des objets JSON à partir de données relationnelles : JSON_OBJECT(), JSON_ARRAY(), JSON_OBJECTAGG(), JSON_ARRAYAGG()

```
-- Nouvelle table relationnelle pour génération d'objets JSON
create table employees ( name varchar2(100), job varchar2(100), wfh
 varchar2(5), address varchar2(100) );

insert into employees (name, job, wfh, address) values ('Loïc', 'PM',
'true', 'Tours, France');
insert into employees (name, job, wfh) values ('Mike', 'Developer', 'false');

-- 2 documents JSON générés (1 pour chaque ligne)
select JSON_OBJECT( 'name' value name, 'job' value job ) from employees;

-- 1 document JSON généré aggrégeant les 2 lignes dans un tableau trié
 par "name"
-- Le champ wfh est converti en valeur JSON booléenne
-- Le champ address est omis dans le document JSON si sa valeur
relationnelle est NULL
-- Ajout du champ calculé "count" correspondant au nombre de lignes /
employés
select JSON_OBJECT( 'allEmployees' value JSON_ARRAYAGG(
                    JSON_OBJECT( 'name' value name,
                                 'job' value job,
                                 'wfh' value wfh format JSON,
                                 'address' value address ABSENT ON NULL
                               ) order by name),
                    'count' value count(*)
                  )
from employees;
```

Figure 23

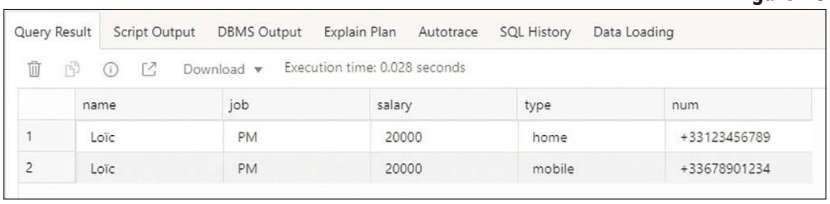
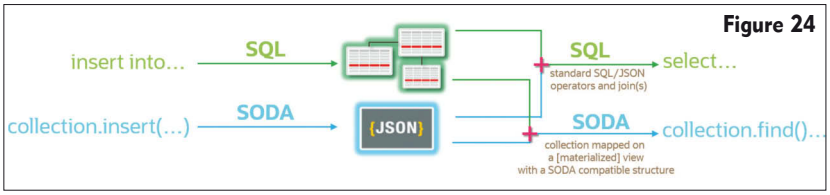


Figure 24



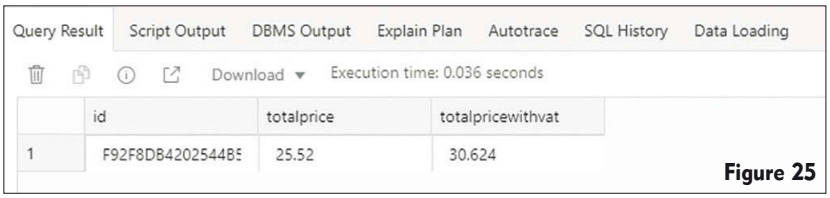


Figure 25

Figure 26

Données JSON et Relationnelles : Fusion ! **Figure 24**

Nous allons utiliser une collection SODA contenant des ordres d'achat et combiner ses documents JSON avec des données relationnelles, ici une table contenant les taux de TVA pour quelques pays. Nous construirons ensuite une seconde collection SODA qui va s'appuyer sur une vue qui elle effectuera une jointure entre les ordres d'achats (JSON) et les taux de TVA (données relationnelles). Une application pourra alors soit interroger directement la vue soit via l'API SODA, récupérer les documents JSON contenant le calcul du prix total de la commande enrichie du prix TTC :

Code complet sur programmez.com & github.

Figure 25

```
-- Récupération d'un document JSON depuis la collection SODA
soda get invoice -f { "country": "France" };
```

Figure 26

Webservices et API REST

Petite cerise sur le gâteau concernant les personnes utilisant une base de données autonome : REST-enabled SQL service. Comme son nom l'indique, c'est la possibilité d'accéder à vos données en SQL au travers d'un endpoint déjà configuré avec le protocole REST. Rappelez-vous lors de la création de l'utilisateur JSONUSER, j'avais indiqué d'activer l'option "REST enable". Ceci a pour effet de donner l'autorisation de se connecter à SQL

Developer web à cet utilisateur, mais cela donne l'accès également à d'autres endpoints. Ainsi, si je veux exécuter une requête SQL, il me suffit de cibler le endpoint du service SQL que l'on peut dériver de l'URL d'accès à SQL developer web :

- endpoint d'accès à SQL developer web :
  https://<database unique name>.adb.<region>.oraclecloudapps.com/ords/<REST alias>/_sdw/?nav=worksheet
- endpoint du service SQL :
  https://<database unique name>.adb.<region>.oraclecloudapps.com/ords/<REST alias>/_sql/

Exemple d'appel via curl et jq pour un rendu plus lisible :

Code complet sur programmez.com & github

La requête SQL cette fois-ci retourne le contenu de la collection SODA nommé invoice. Le retour de l'appel curl est un document JSON contenant le résultat de la requête SQL dans le tableau items. D'autres informations sont également retournées telles que la requête originelle ou encore le type des données. C'est ce point d'accès unique et cette capacité à solliciter en SQL ou PL/SQL la base de données autonome ou sa version on-premises via le composant Oracle REST Data Services (ou ORDS) qui apporte de nouvelles perspectives aux développeurs. Vous avez dit GraphQL ? Oui, cela y ressemble beaucoup…

Base de données convergée

Nous venons de voir un exemple assez simple, mais qui illustre les capacités à faire coexister ces deux mondes Objets/JSON et Relationnels très facilement tout en préservant le choix de l'API pour manipuler ses données : SODA et/ou SQL.

L'éventail de fonctionnalités qui devient accessible aux données JSON est alors fabuleux :

- Collections partitionnées,
- Collections immuables (basée sur les Blockchain tables),
- Fonctions analytiques,
- Column Store (traitement vectoriel en colonnes),
- Vues matérialisées rafraîchies à la demande, au commit ou à l'exécution de requêtes SQL,
- Compression additionnelle,
- Indexation Full-Text,
- Transactions ACID sans aucune limitation,
- web Services / REST API,
- Recherches spatiales via GeoJSON,
- …
- mais aussi combinaison avec des données de type Graph, XML… qu'elles soient stockées en base de données ou à l'extérieur : stockage objet Oracle, AWS, GCP, Azure…, HDFS, NFS, Open Data…

Une dernière nouveauté ?

Avant de clore, je voulais également vous faire part d'une nouveauté disponible avec la dernière version 21c : un nouveau type d'index nommé **Multi-Value Index** (MVI). Le MVI est destiné à indexer les valeurs d'un tableau JSON ou les champs d'objets contenus dans un tableau.

Exemple :

```
-- Création d'une table avec type natif JSON (21c+)
create table jsontab (a number, doc JSON);

-- Insertion de 2 documents contenant des tableaux d'entiers
insert into jsontab VALUES (1, '{"credit_score": [710, 720, 710, 730]}');
insert into jsontab VALUES (2, '{"credit_score": [750, 730, 750, 750]}');

-- Création d'un Multi-Value Index (MVI)
create MULTIVALUE index jsontab_idx
on jsontab t ( t.doc.credit_score.number() );

-- Comptage du nombre de documents contenant au moins une fois la
 valeur 750
-- dans le tableau optionnel credit_score.
select count(*)
 from jsontab t
 where JSON_EXISTS(t.doc, '$.credit_score?(@ == 750)');
```

Figure 27

Le temps d'accès est bien évidemment réduit par rapport à l'approche 19c consistant à créer soit :

- Une vue matérialisée (JSON_TABLE) avec un index b-tree sur la colonne souhaitée,
- Un index Full-Text qui peut potentiellement nécessiter d'indexer tout le document

Plus simple, plus performant et nécessitant moins d'espace disque, le MVI est très intéressant.

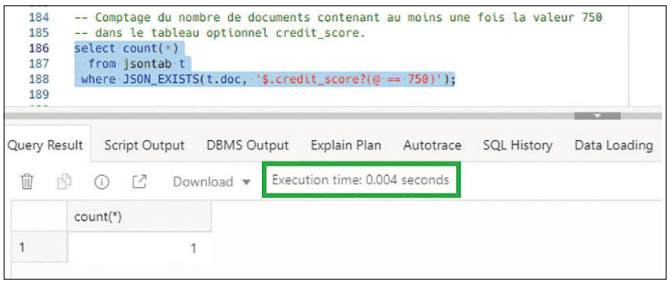
Conclusion

Nous venons de voir au travers d'exemples ce que signifie une base de données convergée en commençant par découvrir les dernières fonctionnalités apportées pour traiter les données JSON quel que soit le type de développeur (frontend : API REST avec point d'accès unique ; backend : API SODA pour JavaScript, Java, Python, PL/SQL, REST… ; business intelligence, data science : SQL et autres capacités à créer des modèles en R ou Python - sans doute un prochain article). Nous avons également vu la force de pouvoir combiner ces données JSON avec des données relationnelles - et cela aurait pu être démontré avec des données de type Graph, XML, RDF. Nous avons également regardé les points forts de ce nouveau type de données natif JSON et ce qui se cachait derrière l'acronyme OSON. Enfin, nous avons découvert ce service managé autonome, le SaaS pour les bases de données qui offre de nouvelles perspectives pour les développeurs que cela soit pour le cycle de vie de la base de données (provisionnement, patching, backup…), la sécurité, les performances avec adaptation dynamique des ressources, les outils web, etc.

Ce service cloud managé et autonome est dorénavant disponible gratuitement sans limite de temps avec 2 bases de données pouvant contenir jusqu'à 20 Go de données, 2 VM… Pour plus de détails : https://oracle.com/cloud/free. Je vous souhaite de belles découvertes sur les LiveLabs Oracle et surtout beaucoup de plaisir à développer vos applications avec cette base de données plus que jamais innovante.

Remerciements : merci à Beda Hammerschmidt et à Savine Schmaltz pour leur aide précieuse dans l'écriture de cet article.

Figure 27



Développeurs : un marché qui redémarre, quid des salaires ?

François Tonic

Après une année 2020 très difficile et un net ralentissement des recrutements, nous constatons depuis 1 an un redémarrage progressif du marché. Le dernier confinement n'a pas cassé totalement la dynamique entamée. Comme toujours, la situation est très différente selon la région. Les modes de recrutement se sont adaptés comme nous l'avons évoqué à plusieurs reprises. Le marché est-il aussi dynamique qu'avant le Covid ? Là encore, attention à ne pas tomber dans les fausses tendances.

Trouver un développeur en 2021 n'est pas plus compliqué qu'en 2020 ou 2019. Bien entendu, il faut nuancer. Celles et ceux qui arrivent sur le marché se focalisent beaucoup sur le développement, notamment JavaScript et tous les frameworks qui gravitent autour. Le développement mobile est concerné, mais dans une moindre mesure. Pour nous, le serveur, desktop et le cloud évoluent dans leur propre sphère. Ces différences de dynamique se comprennent par les nombreuses formations, plus ou moins longues (de quelques mois à 12 mois), qui tournent essentiellement autour du développement web. Les profils de développeurs web sont, comme souvent, les plus faciles à trouver.

Attention : nous parlons de profils débutants. Les profils confirmés et experts sont toujours plus difficiles à trouver. En période de crise, ces derniers ont tendance à rester à leur poste. Ils bougeront en cas d'événements importants ou de propositions motivantes. Le salaire n'est pas le critère premier pour les faire bouger.

Résumé

Depuis quelques semaines, les études se multiplient sur les salaires, comme chaque année. Nous en avons pris deux en référence.

	Urban Linker (étude 2021)			Silkhom		
	0-2 ans	2-5 ans	+5 ans	0-2 ans	2-5 ans	+5 ans
Python	33	43	48	35-45	45-60	60-80
.Net	33	43	48	35-45	45-55	55-75
Java	31,5	40	48	34-43	38-55	42-65
WinDev	-	-	-	35-40	38-55	+45-55
Cobol	-	-	-	32-45	38-50	+45-50
Full JS	36,5	43,5	52	32-48	38-65	43-75
Front-end	33,5	41	51	29-43	35-50	42-58
PHP	31	39	46	27-45	30-55	40-65
Dév mobile	34	41	47,5	32-45	40-55	45-65
DevOps	36	44,5	54	35-50	45-60	+52-60

En k€ / brut annuel

Pour Silkhom : les extrêmes sont élevés car leur étude segmente entre Paris, les grandes villes et les régions. Il est difficile de donner une moyenne. Les extrêmes élevés concernent l'Île-de-France.

Des aberrations qui se multiplient

Aujourd'hui, il est parfois difficile de comprendre la définition de tel ou tel profil technique. Les entreprises et les recruteurs segmentent tellement le métier de développeur que l'on s'y perd. Ainsi, le développeur web se retrouve dans plusieurs profils : fullstack JS, dév JS, back-end, front-end. Si fondamentalement, il est logique de segmenter le développement web, il ne faut pas aller dans l'excès comme actuellement. Car nous n'y comprenons plus grand-chose : la définition de certains profils devient peu claire ou tellement large que l'on ne sait plus très bien ce que le développeur doit faire. La notion de fullstack reste à la mode et permet d'y mettre tout et n'importe quoi. Mais surtout, les études montrent que le fullstack reste relativement mal payé alors qu'il/elle est censé faire tout et n'importe quoi.

Le développeur mobile est une autre aberration. Si nous prenons les contours de Silkhom et d'Urban Linker, il doit connaître Java, Kotlin, Objectif-C, Swift, Go, Cordova, le cross-platform, Flutter sera un bonus. Et on ne peut pas dire que le salaire moyen fasse rêver. En quelques années, le dév mobile est passé de star à un développeur banal. Problème : un bon développeur mobile n'est pas si facile à trouver. Le développeur mobile qui maîtrise Java – Kotlin – Objective-C et Swift est déjà difficile à trouver. Alors rajouter l'hybride avec Flutter et d'autres technos, c'est pour nous un non-sens et le risque de recruter un développeur qui connaît ce que l'on demande, mais sans forcément les maîtriser.

Si je développe un peu en Flutter, est-ce que je suis pour autant un développeur mobile ?

Ingénieur DevOps, ou simplement DevOps, voilà un bien étrange terme pour un profil. Voici la définition de l'objectif de Silkhom : « créer des applications qui sont parfaitement adaptées à l'infrastructure (et qui continuent à fonctionner une fois mises en place) ».

Nous l'avons lu et relu, nous ne comprenons rien à cette définition. Que signifie donc ingénieur DevOps ? Rappelons que le DevOps est une philosophie et des bonnes pratiques. Les compétences autour du DevOps sont multiples et se fondent dans les équipes et les profils techniques.

Le Cobol intéresse de moins en moins les études salariales IT. On le constate ici : seul Silkhom le mentionne en confirmant la tendance de ces dernières années qui indique que ce langage paie mal. Or c'est dommage : d'un côté le développeur Cobol est une ressource de plus en plus rare, à fortiori avec de l'expérience, de l'autre ce langage est encore au cœur de l'activité de bon nombre d'entreprises. Alors certes, le Cobol n'est pas attirant, mais le dévaloriser ne va pas favoriser la formation donc le recrutement de nouveaux Cobolistes! Ce

> « Arrêtons de dévaloriser le Cobol »

n'est pas un langage attirant, mais le dévaloriser ne va pas favoriser la formation de nouveaux Cobolistes.

Quelles dynamiques ?

Talent.io a publié une étude sur les salaires il y a quelques semaines. D'après leurs données, les compétences / les profils les plus demandés selon les villes sont :

Villes	Compétences / profils
Paris	Fullstack Back-end Front-end
Bordeaux	Fullstack Lead Dev Back-end
Lyon	Fullstack Back-end Front-end
Toulouse	Fullstack Architecte logiciel Front-end
Lille	Fullstack Back-end Lead dev / front-end

Selon Talent.io, les demandes sont assez homogènes dans les grandes villes ce qui n'est pas forcément une surprise. Bien entendu, les spectres sont larges, on peut donc y mettre beaucoup de technologies et de langages.

L'étude met aussi en évidence une chute importante du nombre d'entretiens par candidat sur le site talent.io : - 50 %. Ce qui montre que 2020 a été fortement impacté. Pour 2021, la tendance s'inverse. Mais il faudra attendre quelques mois pour retrouver le niveau ante-Covid.

Le site rappelle aussi, à juste titre, une stagnation, voire une légère baisse, des salaires moyens, au moins, les développeurs peu expérimentés : -2-3 % sur Paris. En régions, il y a une légère hausse sur les profils débutants, mais une baisse globale. Cependant, ces variations se font à la marge et elles disparaîtront rapidement.

Comme toujours, nous constatons sur les salaires une différence nette entre Paris et le reste de la France : jusqu'à 20 % sur certains profils.

Selon les données de talent.io, les technos les plus recherchées sont :

Domaines	Technologies / langages
Backend	Node, PHP, Python, Java
Frontend	React, JS, Vue, Angular

Tendances 2021-2022

Clairement, le marché de l'emploi IT, et particulièrement du développement, retrouve sa dynamique. Nous le constatons depuis septembre 2020 et a tendance à accélérer depuis le printemps 2021. Il va rapidement retrouver le niveau de 2019. Il faudrait une nouvelle crise sanitaire majeure pour casser la dynamique, mais l'impact devrait être moindre qu'au printemps – été 2020, car les entreprises se sont, globalement, adaptées.

De nombreux nouveaux arrivants, nouveaux diplômés, reconversions et formations courtes, ont subi 2020. Cela signifie un nombre plus élevé de profils débutants ce qui s'observe aussi par des salaires moyens inchangés ou en légère baisse. Par contre, la situation restera tendue pour les profils expérimentés et les compétences pointues. En période de crise / défavorable, la tendance est de rester à son poste.

Pour les free-lance qui ont subi une claque au 1er confinement, la situation s'améliore depuis l'été / la rentrée 2020. Les indépendants peuvent pallier aux faibles recrutements. Il faudra surveiller le tarif journalier, indicateur de la bonne santé, ou non, du free-lance.

N'hésitez à nous envoyer vos retours / commentaires.

Nous remercions Océane Roudier pour sa relecture et ses conseils.

USA : TÉLÉTRAVAIL = SALAIRE EN BAISSE

Est-ce un cas isolé ou le début d'une tendance ? L'affaire fait du bruit : Google pourrait baisser les salaires des employés qui optent pour 100 % de télétravail et qui habitent en dehors de la région de San Francisco ou des villes chères. Selon Reuters, la baisse pourrait atteindre 25 % dans certains cas, avec une moyenne de 10-15 %. Chez les géants de la valley, des salariés veulent plus de télétravail et quand ils veulent dans la semaine, c'est le cas d'une pétition lancée chez Apple. Des salariés estiment que la direction n'est pas assez souple.

Ce mouvement d'ajustement s'observe depuis la fin 2020 chez VMware, Twitter, Reddit, Stripe, Facebook…

PROGRAMMEZ!

Programmez! n°248
septembre - octobre 2021

Directeur de la publication & rédacteur en chef
François Tonic
ftonic@programmez.com

Contacter la rédaction
redaction@programmez.com

Ont collaboré à ce numéro
Louis Adam (ZDnet.fr)

Code review :
Dorra Bartaguiz

Les contributeurs techniques

Jérémy Favier
Stéphane Cadot
Grégory Bersegeay
Paul Peton
François Genestin
Maxime Billemaz
Vincent Thavonekham
Denis Duplan
Wassim Chegham
Chihab Otmani
Sun Tan
Yohan Lasorsa
Arnaud Thiefaine
Jimmy Tran
Philippe Martin
Sallah Kokaina
Jean-Christophe Riat
Marc Hage Chahine
Loïc Lefèvre

Maquette
Pierre Sandré

Marketing – promotion des ventes
Agence BOCONSEIL - Analyse Media Etude
Directeur : Otto BORSCHA
oborscha@boconseilame.fr
Responsable titre : Terry MATTARD
Téléphone : 09 67 32 09 34

Publicité
Nefer-IT
Tél. : 09 86 73 61 08
ftonic@programmez.com

Impression
SIB Imprimerie, France

Dépôt légal
A parution

Commission paritaire
1225K78366

ISSN
1627-0908

Abonnement
Abonnement (tarifs France) : 49 € pour 1 an, 79 € pour 2 ans. Etudiants : 39 €. Europe et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € - Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 €.
Autres pays : consultez les tarifs sur www.programmez.com.
Pour toute question sur l'abonnement :
abonnements@programmez.com

Abonnement PDF
monde entier : 39 € pour 1 an.
Accès aux archives : 19 €.

Nefer-IT
57 rue de Gisors, 95300 Pontoise France
redaction@programmez.com
Tél. : 09 86 73 61 08

softfluent

Rejoignez **l'ESN** créée par des **développeurs** pour les Développeurs

Vous possédez une ou plusieurs de ces compétences ?

- Azure DevOps
- Cloud
- SQL Server
- Angular
- React
- C#
- CI/CD
- .NET Core
- ASP.NET MVC
- Microservices

Contactez nous !

Retrouvez-nous sur notre page carrière : **softfluent.fr/nous-rejoindre**

Ou contactez Marine, en charge du recrutement chez SoftFluent :

06 69 26 27 87

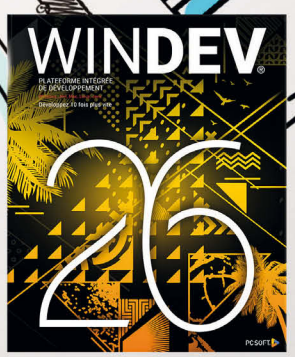
marine.genetay@softfluent.com

- Conseil
- Expertise
- Partage

softfluent.fr

TRANSFORMEZ VOS GRANDES IDÉES EN LOGICIELS

C'EST FACILE AVEC WINDEV 26



PCSOFT.FR