PROGRAMMEZ!
N°246
05/06
2021

PROGRAMMEZ!

Le magazine des développeurs

PyQt : Quand Qt rencontre Python

TensorFlow

Gérer du code Legacy

Recrutement : un rebond ?

+5 000 postes ouverts

Le TDD dont vous êtes le héros

Générer la doc avec DocFX

Refactoring

IoT par la pratique

Le seul magazine écrit par et pour les développeurs

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

M 04319 - 246 - F: 6,99 € - RD

DÉVELOPPEZ 10 FOIS PLUS VITE

WINDEV Tech Tour 26

926 NOUVEAUTÉS

NOUVELLE VERSION

REGARDEZ LES REPLAYS

CETTE ANNÉE, LE **WINDEV TECH TOUR** S'EST DÉROULÉ
SOUS FORME DE 3 WORKSHOPS EN LIGNE

REGARDEZ LES REPLAYS DES WORKSHOPS SUR
pcsoft.fr/windev/videos.htm

WWW.PCSOFT.FR

WINDEV, AGL DEVOPS LOW CODE

Contenus

Divers

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

EDITO#246*

Un jour sans fin

© Excelsior, 17 février 2021. Tournage du Project X

Les dernières semaines furent mouvementées et pas seulement à cause de notre ami, Covid. Le procès sans fin Oracle contre Google sur l'utilisation de codes Java dans Android a été définitivement jugé, ou presque.

Pour faire simple : Google a gagné. Grosso modo, Google n'a pas violé les droits d'auteur en copiant l'API Java. Google gagne sur sa défense : les API ne peuvent être protégées par le droit d'auteur et même si c'était le cas, bah, Google pouvait tout de même le faire en invoquant l'usage loyal. Merci à toi Fair Use. Mais finalement, la cour suprême américaine ne tranche pas la question de fond et qui est pourtant aussi importante que le procès en lui-même : est-ce que les API peuvent être protégées par le droit d'auteur, et recevoir un joli © ?

Finalement, ce jugement risque de ressembler à un jugement à la Ponce Pilate dans le sens où le juge refuse de porter la responsabilité de se positionner sur la question fondamentale de l'opposition Oracle – Google. Et on ne sait pas réellement quelles sont les limites du fair use sur les API, ni ce qui peut être soumis au droit d'auteur ou non et si cela concerne l'ensemble des API ou seulement certaines.

Vous avez 404 ans pour en débattre.

FLUTTER 2

La grosse sortie du mois est Flutter 2.x. La plateforme permet de générer des apps pour mobile, web et desktop. Sur la partie desktop, la version stable arrivera sans doute fin de l'année, mais c'est une évolution majeure. Désormais, Flutter couvre la plupart des usages. Surtout, Flutter met une pression sur les autres plateformes, on pense bien entendu à Xamarin qui souffre d'un manque de visibilité et d'une roadmap claire.

Je suis impatient de voir comment Flutter va évoluer dans les prochains mois. J'attends un support complet sur les architectures ARM pour pouvoir développer sur n'importe quelle poste. Comme toujours, Flutter doctor est ton ami.

Une des questions sera de voir si Flutter se détachera de Google pour devenir un projet « indépendant ».

OPENJDK SAUCE MICROSOFT

Microsoft a mis à disposition des développeurs et entreprises sa propre version d'OpenJDK. Cette distribution repose tout naturellement sur le projet open source. Il existe de multiples JDK sur le marché. Cette première version supporte Java 11. La prochaine release sera basée sur Java 17.

L'éditeur a fait un gros effort pour porter la JDK sur AArch64. Par contre, aucune nouvelle précise de la version Apple Silicon que Microsoft avait évoquée à l'automne 2020.

Mi-avril, les équipes Visual Studio annoncèrent un petit événement : la version 2022 sera la première version 64 bits ! Eh oui, on oublie souvent que Visual Studio est toujours une app 32 bits. Faites chauffer la RAM !

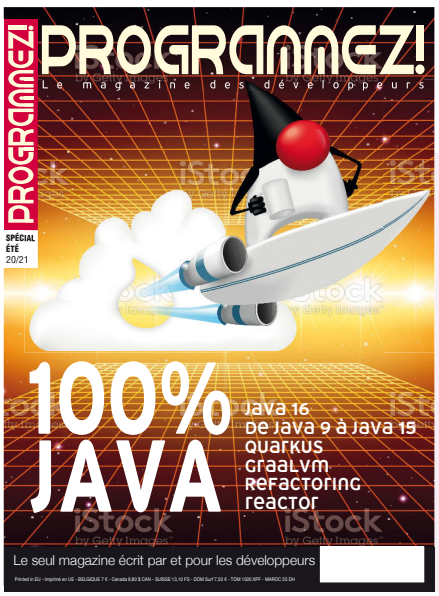
FAITES LE STOCK DE CAFÉ !

Le printemps est toujours difficile pour le développeur. En 3 semaines, les trois conférences les plus attendues se succèdent à un rythme infernal : Google I/O, Microsoft BUILD et enfin la WWDC d'Apple !

On s'attend à une avalanche de démos, de nouveautés et d'annonces. Comme chaque année, Programmez! vous proposera des lives, des comptes-rendus.

En attendant, Programmez! est bon pour vos neurones.

**En réalité le n°249*



LES PROCHAINS NUMÉROS

HORS SÉRIE #4 ÉTÉ

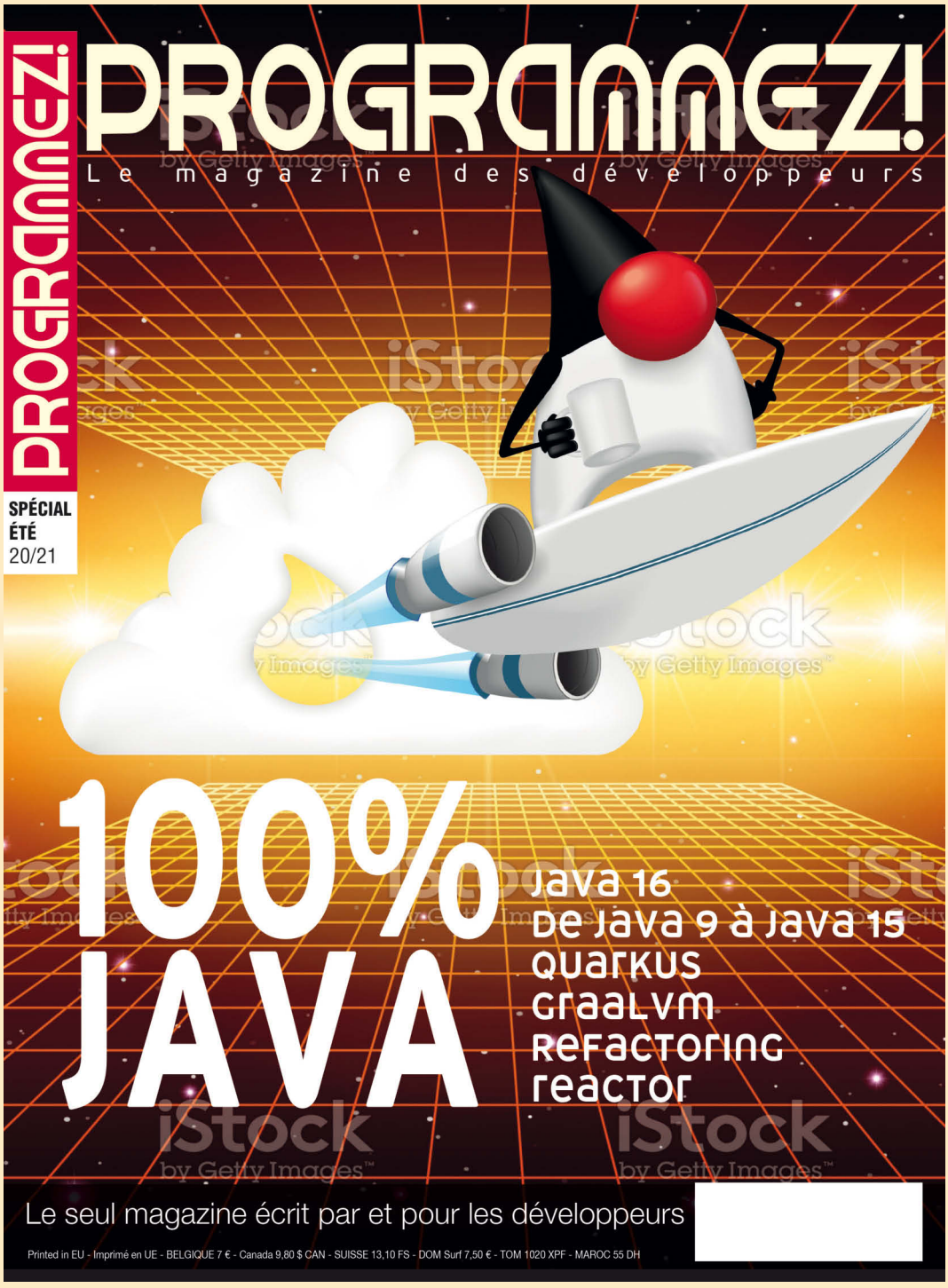
100 % JAVA

Disponible le 9 juillet

Programmez! n°247

Spécial Maker / IoT / Coding4fun

Disponible le 2 juillet



100% JAVA

disponible dès le 9 juillet 2021

Kiosque / Abonnement - Version papier / Version PDF

Attention : agenda pouvant changer à tout moment selon les interdictions.

Les événements Programmez!

Meetups Programmez!
29 juin : sujet à venir
Où : virtuel pour le moment
A partir de 18h30

DevCon by Programmez !
Septembre : .Net 2e édition

Informations & inscription : programmez.com

mai

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
	Devopsdays Paris					
	Google I/O					
24	25	26	27	28	29	30
	Microsoft BUILD			AFUP Day Lille / Rennes		
31						

juin

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
	1	2	3	4	5	6
7	8	9	10	11	12	13
				AFUP Day Toulouse / Tours		
14	15	16	17	18	19	20
BlendWebMix (Lyon)			Blazor Day			
21	22	23	24	25	26	27
VoxxedDays / Luxembourg						
28	29	30	1 juil.	2 juil.		
		Devoxx France				

juillet

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
			1	2	3	4
			Devoxx France			
				Nantes Maker Campus		
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

RivieraDev annonce sa journée développeur début juillet à Sophia Antipolis. A l'heure où nous imprimons, pas de précision de date.

POUR LA RENTRÉE !

- **JFTL :** 13 & 14 septembre à Montrouge (près de Paris)
- **Big Data Paris :** 28 & 29 septembre à Paris
- **DevFest Nantes 2021 :** 21 & 22 octobre à Nantes
- **Hack in Paris :** du 15 au 19 novembre
- **DevFest Lille 2021 :** 19 novembre à Lille

SANS DATE

- MixIT
- NCrafts Paris
- BreizhCamp
- DevFest du bout du monde
- RivieraDev
- Best of Web
- Flutter Con Paris
- Sunny Tech
- DevFest Toulouse
- DevFest Nantes

RENDEZ-VOUS EN 2022

DevOps Rex
Sunny Tech

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

LA MANUFACTURE CACD2

Niveau padawan

OSAXiS

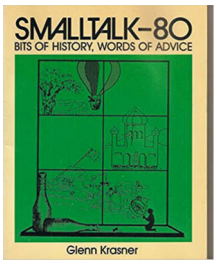
Vous voulez soutenir activement Programmez! ?
Devenir partenaires de nos dossiers en ligne et de nos événements ?

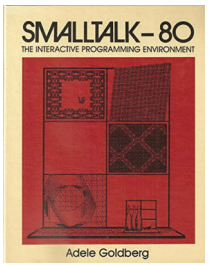
Contactez-nous dès maintenant :

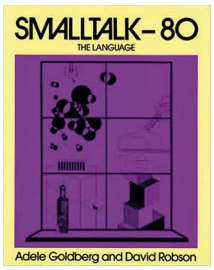
ftonic@programmez.com

Smalltalk : le retour d'un langage mythique

PARTIE 3



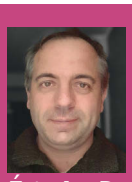




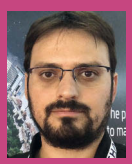
Terminons notre voyage au coeur de Smalltalk avec la 3e et dernière partie. Nous alternons retour terrain et technique.

La rédaction

Dossier relu et coordonné par **Laurent Julliard** *et* **Stéphane Ducasse.**



Éric Le Pors



Pierre Laborde

THALES

Smalltalk chez Thales

Smalltalk est partout et notamment dans l'industrie. Nous avons voulu en savoir avec Éric Le Pors, UX & UI Lead chez Thales DMS France à Brest et Pierre Laborde, Senior UX (User eXperience) & UI (User Interface) Designer chez Thales DMS France à Brest. Thales est un leader mondial dans l'industrie de l'électronique de défense. Thales Defense Mission Systems est une entreprise du groupe Thales, c'est une entité regroupant des activités de fournisseur de systèmes et d'équipements dans les domaines de la surveillance et reconnaissance maritime, la guerre électronique, les radars, etc.

Comment êtes-vous venus à Smalltalk sur un plan personnel ou professionnel et qu'est-ce qui vous a conquis ?

Éric : J'ai rapidement fait connaissance avec Smalltalk enseigné très tôt dans le cursus pour faire découvrir les principes objets aux étudiants. J'ai tout de suite été impressionné par la puissance du langage et la facilité de créer des programmes d'une façon totalement nouvelle. Si on ne se laisse pas dérouter par sa syntaxe, on plonge alors dans un monde où tout est objets et messages. La première chose que l'on constate c'est que tout le code utile pour son programme se trouve dans « l'image », un ramasse-miettes s'occupe de collecter les objets inutiles et de les détruire pour nous ! On peut même coder directement dans le débogueur et effectuer dynamiquement des modifications en fonction des entrées disponibles sur la pile d'exécution. On peut modéliser directement une hiérarchie d'objets réifiant des concepts complexes, puis les utiliser ou les modifier dynamiquement et à volonté.

C'est notamment pour toutes ces raisons que nous avons choisi d'utiliser Smalltalk chez Thales pour le maquettage. En effet, c'est sous l'impulsion de mon ancien mentor Vincent Verbeque, qui avait perçu les capacités naturelles de ce langage lors des phases de prototypage, que nous avons commencé à utiliser Smalltalk pour créer rapidement de petites maquettes rapidement modifiables. Aujourd'hui nous l'utilisons même pour réaliser des démonstrateurs complexes.

Ce qui me séduit vraiment dans le langage Smalltalk, c'est sa grande productivité, sa flexibilité et sa malléabilité : avoir la capacité de changer dynamiquement l'arbre d'héritage d'une classe, ajouter une variable d'instance ou de classe à la volée... Cette flexibilité a toutefois un revers : ses méthodes et variables n'utilisent pas de type et un programme peut dans l'absolu lever une exception durant l'exécution (ndlr: les langages typés ne sont pas exempts de ce genre de problème...)

Ces dangers ont été pour moi formateurs et m'ont donné une grande rigueur dans l'écriture de mes programmes. Quand on a goûté à cette flexibilité, aux facilités qu'elle procure, on devient rapidement « fan » de ce langage et on a du mal ensuite à revenir à autre chose.

Faire du prototypage IHM en Smalltalk c'est extrêmement puissant : imaginez modifier en séance, devant votre client et en quelques lignes de code, modifier l'apparence, l'organisation visuelle ou les animations de vos IHM et ceci sans arrêt pour recompiler ou sans recharger une page, directement en modifiant des valeurs depuis un point d'arrêt dans le débogueur par exemple. En quelques minutes, créez devant votre client une forme graphique décrite en objet, assemblez-la avec d'autres formes, colorez-la et vous affichez désormais un interacteur innovant immédiatement utilisable... comment ne pas être conquis ?

Pierre : À l'origine je viens du monde Web (Adobe Flash, PHP, JavaScript, etc.), j'ai appris Java en entrant chez Thales afin de pouvoir réaliser un premier prototype IHM. Je voulais proposer des interfaces graphiques plus organiques (c. à.d. moins basées sur des matrices de widgets alignés en colonne et en lignes comme ce qui est fait classiquement avec Swing), notamment pour proposer des interfaces tactiles innovantes.

Dans ce but, nous avons développé une surcouche à Java2D pour pouvoir réaliser ce type d'interface plus facilement, sorte de JavaFX avant l'heure. Puis j'ai découvert les capacités de Smalltalk (sous Cincom VisualWorks) grâce à Éric Le Pors qui commençait aussi à prototyper des interfaces graphiques, nous avons commencé à travailler ensemble. Même si les outils de conceptions étaient rudimentaires nous avons décidé de basculer tout le prototypage en Smalltalk afin de continuer la réalisation du prototype IHM Java que j'avais initié précédemment. Nous avions décelé un potentiel énorme

du Smalltalk pour le prototypage IHM. Au tout début ce n'était pas facile pour moi, mais il ne m'a fallu que quelques semaines pour m'habituer à cette nouvelle façon de programmer et j'ai pu avec le temps me défaire de ma précédente manière de programmer sur des éditeurs classiques (comme Eclipse), l'éditeur Smalltalk étant très intuitif. Nous avons écrit rapidement des outils et des surcouches graphiques adaptés au prototypage en nous reposant sur les bonnes propriétés du langage.
Les capacités graphiques de VisualWorks étant quasi inexistantes ou obsolètes nous avons écrit de nouvelles couches plus modernes en utilisant des librairies externes de dessin (comme CairoGraphic) et en nous basant sur le tout objet de Smalltalk pour obtenir des capacités complètement dynamiques. Je pense que mon expérience dans les API graphiques et dans le design d'IHM nous a aidés à faire les bons choix pour concevoir des outils orientés designers et conception centrée utilisateur et non des outils pour les ingénieurs en programmation. La simplicité et l'universalité de Smalltalk nous permettent de réduire les compétences en programmation de nos équipes de prototypage pour nous concentrer sur le design IHM.
Je trouve Smalltalk extrêmement facile à prendre en main, il bouscule vraiment les codes de la programmation classique alors que, paradoxalement, c'est un langage qui a vu le jour il y a plus de 40 ans ! Pour moi c'est un excellent mélange entre le caractère flexible et dynamique des langages Web et le caractère robuste et optimisé des langages natifs, en effaçant les défauts de chacun ! Je pense que ce qui me plaît le plus c'est la fluidité, pas d'un point de vue performances (ndlr : la machine virtuelle de Pharo est toutefois très performante), mais d'un point de vue workflow, je ne suis jamais bloqué ni perdu, on avance vite et on retombe toujours sur ses pattes en Smalltalk, je trouve qu'il n'y a pas de superflu c'est ce qui fait que l'on va très vite pour faire des choses. Un seul fichier image porte à la fois l'environnement de développement, d'exécution, les ressources, etc. Ce qui est génial c'est que tout bout de code peut être testé en l'exécutant n'importe quand juste en effectuant un « Do It » dessus.
Autre différence majeure par rapport à d'autres langages : Smalltalk fonctionne avec un fichier image qui contient toute l'application et son éditeur. Lorsque vous enregistrez votre image Smalltalk, vous sauvegardez aussi tout son contexte d'exécution. Ce n'est pas seulement le code source, mais aussi toute l'empreinte mémoire qui est stockée dans l'image Smalltalk. Lorsque vous rouvrez votre image, vous retrouvez votre environnement tel que vous l'aviez laissé, les fenêtres graphiques ouvertes, toutes les instances d'objets créés précédemment, les threads, etc. L'image Smalltalk c'est comme une recette de cuisine avec tous les ingrédients nécessaires : vous trouvez les instances d'objets, le code source et toutes les ressources annexes nécessaires à l'application (photos, données, etc.). C'est possible de sauvegarder une image Smalltalk suite à un plantage sur le poste d'un client (par exemple ouverture du débogueur suite à une exception), envoyer l'image sur un poste de développement pour continuer le débogage et l'analyse tranquillement. J'ai aussi été conquis par l'environnement de programmation, la navigation est très intuitive et la lecture du code n'est jamais offusqué, pour moi c'est un environnement de modélisation très efficace, sans oublier la modification intégrale du code : de la valeur des variables d'instances à la structure des classes et tout ça sans jamais arrêter l'exécution de l'application.
En prototypant en Smalltalk on se surprend souvent à développer avec l'application qui tourne, c'est comme un arbre qui s'étend : on construit les branches sans interrompre la pousse, on identifie les parties du code manquantes ou non fonctionnelles grâce aux fenêtres du débogueur qui s'ouvrent, on complète en adaptant le code, le cassant parfois, construisant des pans entiers du tronc, tout en continuant l'exécution de l'application...

Quel rôle joue Smalltalk dans les produits ou services de Telna (ou de Thalès) et quelles sont les fonctionnalités de Smalltalk qui vous ont été les plus utiles ?

Pierre : Nous avons développé en interne des outils de conception qui se basent directement sur les propriétés du langage, c'est notre framework de prototypage appelé « Smock » (pour Smalltalk MOCKup). Nous sommes en train d'en développer une nouvelle version en open source sur Pharo (projet open-Smock), car Pharo est un Smalltalk qui remet le langage au goût du jour sur beaucoup d'aspects, notamment les IHM. Smalltalk nous fait gagner un temps considérable à chaque itération de conception, c'est pourquoi nous pouvons non seulement aller très vite, mais aussi proposer beaucoup plus d'itérations de conception pour faire maturer des solutions, là où dans d'autres langages plus classiques nous sommes freinés par des lourdeurs ou des contraintes.
Comme c'est un langage entièrement dynamique, il nous permet de faire évoluer nos prototypes et de pouvoir construire par itération des solutions à la volée avec les utilisateurs. Imaginez un utilisateur devant son poste de travail, il déroule un scénario de travail sur l'un de nos postes, nous pouvons concevoir avec lui directement des adaptations testées à la volée sans avoir à relancer l'application, ni faire un refresh, ni fermer tout son contexte de travail. Tout est transparent, l'utilisateur n'est pas perturbé par de la technique logicielle et au contraire s'implique naturellement dans la conception, car il voit les changements directement devant lui. Nous testons des itérations de la solution (mesures physiologiques à l'appui) avec lui jusqu'à l'obtention d'une solution qui fait consensus (modifier l'ordre des éléments d'un menu, ajouter des raccourcis, modifier les éléments de texte affichés, les tailles et les couleurs, etc.). Le Smalltalk nous offre ces capacités sur un plateau.
Le débogueur Smalltalk est une fonction maîtresse sur laquelle nous nous appuyons en permanence. Il nous permet de réduire considérablement notre temps de conception et de traiter dynamiquement tous les problèmes que nous n'avons pas pu identifier. Lorsqu'un « plantage » arrive lors de l'exécution de l'application il est intercepté, analysé et corrigé à la volée par nos développeurs grâce aux outils du débogueur et appliqué à l'ensemble des stations de travail pour continuer le déroulement du scénario comme si de rien n'était, c'est une opération qui impressionne toujours nos utilisateurs : ils ont l'habitude de devoir tout refaire suite à des bogues.
La force du debugger c'est qu'il est possible de naviguer dans l'exécution du code comme on le souhaite, de revenir en arrière, d'aller en avant, de faire des sauts dans les instructions et de forcer des exécutions. Il permet de sonder (point d'arrêt) la modification d'une variable particulière d'une instance de classe particulière et c'est très utile en IHM.

Éric : Nous avons développé Smock avec Pierre bien au-delà de ce pour quoi il était prévu à l'origine. D'un outil simple et efficace pour créer des maquettes IHM, nous en avons fait un atelier capable de prototyper des systèmes toujours plus complexes, interconnectés avec d'autres langages (C, Java, Python…) et d'autres environnements comme la 3D, des senseurs embarqués…
Les principales propriétés de Smalltalk qui ont permis cela sont évidemment les capacités de Debug que citait mon collègue Pierre, mais aussi les capacités réflexives du langage : c'est à dire la capacité que l'on a depuis le code lui-même d'aller chercher son propre état, de monitorer la pile d'exécution, d'altérer même jusqu'à son propre fonctionnement.
Une autre fonction utile dans le langage, c'est sa connexion aux primitives en C par exemple. Cette capacité rend en effet le langage connectable à à peu près tout et n'importe quoi… Dessiner avec du Cairo Graphics, du SDL, effectuer des opérations sur le window manager, le système de fichier, afficher des flux vidéo, piloter dynamiquement leur affichage depuis une zone de texte…
Le texte, le code, c'est cela aussi sa force… cette capacité à écrire un petit bout de code, de le surligner avec la souris et de demander son exécution immédiatement, depuis à peu près n'importe où dans le langage : dans un espace dédié « Playground », dans un éditeur de code, dans le débogueur, cela rend le langage dynamiquement testable. C'est une sorte de bac à sable géant pour développeur. Par exemple, le code « 5 seconds wait » en Pharo va simplement vous permettre d'effectuer une pause de 5 secondes. Vous pouvez surligner uniquement ce code et demander au langage de le faire ou vouloir inspecter la valeur de retour voire même d'aller regarder comment marche le code lui-même via un débug. On découvre alors parfois dans le langage lui-même des implémentations hyper-élégantes, des design patterns à répliquer encore et encore… Car parmi une des propriétés géniales de ce langage c'est que l'intégralité du code smalltalk de l'image est accessible et déboguable. Ce qui laisse accessible à nos jeunes développeurs un ensemble de pattern logiciels qui, au fur et à mesure, vont contribuer à améliorer leur pratique de développement.

Sven Van Caekenberghe

Adore le développement pour le cloud et les systèmes web en utilisant des langages dynamiques de haut niveau tels que Smalltalk ou LISP. Il est cofondateur de Beta Nine, une société d'ingénierie située à Hasselt en Belgique. C'est un fier supporteur de la Pharo Association et de son consortium industriel dont il est membre du comité de décision. Il développe et maintient activement des bibliothèques clés de Pharo.

Mettez un peu de HTTP dans Pharo

DES EXEMPLES RÉELS DE CONCEPTION OBJET

HTTP est l'une des plus importantes technologies sous-jacentes d'Internet. Tout système de programmation moderne se doit de proposer une bonne implémentation de HTTP. Mais un système Smalltalk demande davantage: son implémentation de HTTP doit être écrite dans le langage Smalltalk lui-même pour être totalement comprise et maîtrisée par ses utilisateurs.

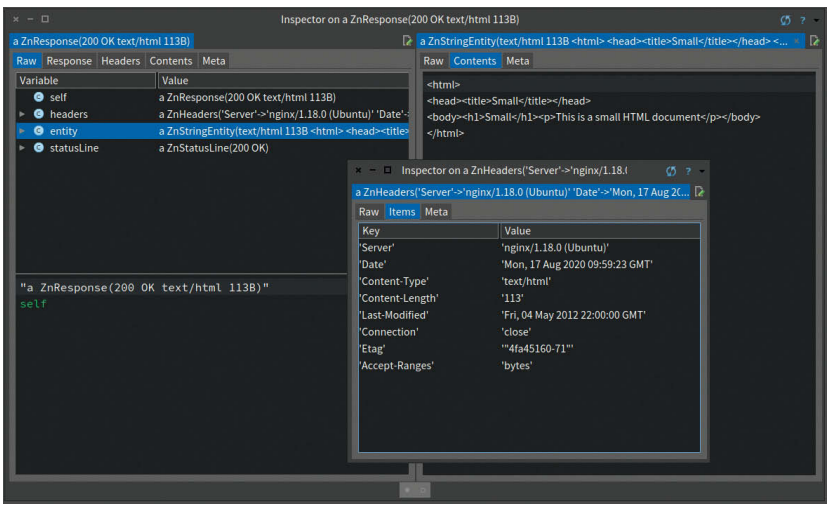
Zinc HTTP components fournit à Smalltalk / Pharo une implémentation du protocole HTTP incluant à la fois un client et un serveur de haut niveau. Il est utilisé comme fondation de nombreuses applications, y compris celles qui consomment ou publient des services Web ou d'autres qui sont bâties au-dessus tel le framework d'applications Web Seaside.
Cet article met en avant un certain nombre de cas de conception objet dans la vraie vie à partir du framework Zinc HTTP Components.

Modéliser le protocole

HTTP est composé de requêtes envoyées d'un client à un serveur et de réponses en retour au travers du réseau. Requêtes et réponses sont assez similaires : elles contiennent toutes les deux des en-têtes (une collection de métadonnées sous forme de clés-valeurs) et une entité optionnelle à savoir la ressource à transférer telles qu'un document HTML ou du texte. Elles diffèrent dès leur toute première ligne: les requêtes commencent par une ligne de request alors que les réponses débutent avec une ligne d'état.
La conception de ZInc découle naturellement de la spécification. Nous créons une superclasse Message avec Request et Answer comme des sous-classes. Une classe Header encapsule la gestion des métadonnées clé/valeur, tandis que les classes Request et StatusLine représentent les premières lignes. Un ensemble de sous-classes d'entités implémente la partie facultative entité / ressource.
Du point de vue du comportement, le protocole nous dit ce que nos objets message doivent être capables de faire. Du côté client, il doit être simple et élégant de créer et de configurer un objet requête. Ensuite, il devrait être possible d'écrire cet objet composite dans un flux réseau. Du côté serveur, l'objet de requête est lu à partir d'un flux réseau et reconstruit dans la même structure. Pour la réponse, un chemin similaire est suivi côté serveur.
Cela nous amène à définir 2 messages clés: readFrom: et writeTo: que tous nos objets HTTP doivent implémenter. Dans un objet de requête composite par exemple, #writeTo: est implémenté en déléguant à sa ligne de requête, des en-têtes et des sous-objets facultatifs.
En utilisant des accesseurs soigneusement sélectionnés et des méthodes de création d'objets côté classe, nous construisons une API pour gérer HTTP avec élégance. Regardons un peu de code. Voici un script qui exécute une requête GET côté client conformément à la spécification du protocole.

```
| url request response connection |
url := 'http://stfx.eu/small.html' asUrl.
request := ZnRequest get: url.
request setConnectionClose.
connection := ZnNetworkingUtils socketStreamToUrl: url.
request writeOn: connection.
connection flush.
response := ZnResponse readFrom: connection.
connection close.
response.
```

Figure 1



Nous définissons d'abord un objet URL. Ensuite, nous utilisons cette URL pour créer un objet de type requête. La méthode de création d'instances get: configure cet objet composite dans ce cas courant. À titre d'exemple, nous envoyons setConnectionClose à la demande pour indiquer qu'il s'agira d'un cycle de demande / réponse unique - cela définit l'en-tête de connexion égal à Close.
Ensuite, nous ouvrons une connexion réseau (flux de socket TCP) vers la partie hôte de l'URL. Nous sommes maintenant prêts à envoyer notre requête en l'écrivant sur le réseau. Enfin, nous vidons le flux pour nous assurer que les données sont entièrement transférées.
Si tout se passe bien, nous pouvons maintenant utiliser la connexion pour lire la réponse du serveur. **Figure 1**

Dans le script ci-dessus, vous pouvez inspecter n'importe lequel des objets impliqués pour déterminer ce qu'ils

représentent. Les représentations imprimées standard de la demande et de la réponse sont déjà utiles.

```
a ZnRequest(GET /small.html)
a ZnResponse(200 OK text/html 113B)
```

À des fins de débogage, nous pouvons également jeter un œil à ce qui s'est passé sur le fil en écrivant la requête et la réponse dans le Transcript.

```
request writeOn: Transcript.
response writeOn: Transcript.
Transcript flush.
```

Ce qui donne la sortie suivante. Vient d'abord la demande:

```
GET /small.html HTTP/1.1
User-Agent: Zinc HTTP Components 1.0 (Pharo/7.0)
Accept: */*
Host: stfx.eu
Connection: close
```

La première ligne est la ligne de requête, les 4 autres lignes constituent l'en-tête, il n'y a pas d'entité. Ensuite, la réponse est affichée.

```
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Thu, 13 Aug 2020 09:29:56 GMT
Content-Type: text/html
Content-Length: 113
Last-Modified: Fri, 04 May 2012 22:00:00 GMT
Connection: close
Etag: "4fa45160-71"
Accept-Ranges: bytes

<html>
<head><title>Small</title></head>
<body><h1>Small</h1><p>This is a small HTML document</p>
</body>
</html>
```

Ici, la première ligne est la ligne d'état, puis viennent 8 entêtes, une ligne vide et l'entité, un document HTML.
Du côté serveur, quelque chose de similaire se produit: la requête est lue, un objet de réponse est construit et écrit. Voici un exemple de la manière dont une telle réponse pourrait être construite.

```
| page |
page := ZnHtmlOutputStream streamContents: [ :html |
 html page: #Small do: [
  html tag: #p with: 'This is a small HTML document' ] ].
ZnResponse ok: (ZnEntity html: page)
```

Cela donne une première idée de ce qui se passe dans les composants HTTP de Zinc et à quoi ressemble une approche basée sur les objets de HTTP.
L'approche de haut niveau consiste à utiliser un objet client HTTP qui a une API de style constructeur: vous construisez et configurez votre requête avant de l'exécuter, le résultat étant stocké dans le client.

```
ZnClient new
 url: 'http://stfx.eu/small.html';
 beOneShot;
 systemPolicy;
 accept: ZnMimeType textHtml;
 get.
```

Pour ceux qui ont déjà lu l'article d'introduction au langage Smaltalk vous noterez ici l'usage du point-virgule comme opérateur de cascade qui permet d'envoyer une série de messages à un même objet.
Le message beOneShot indique au client d'agir comme nous l'avons fait précédemment, de n'exécuter qu'un seul cycle de demande / réponse. Le message accept: spécifie le type (mime) de ressource (entité) que nous voulons. Le message systemPolicy demande au client de s'assurer à la fois de la réussite HTTP et d'un type mime correspondant ou de signaler une erreur. Le résultat du message get est le contenu réel de la ressource, dans ce cas le HTML sous forme de chaîne. Si nécessaire, l'objet de réponse complète reste disponible dans l'objet client. Au fait, le moyen le plus court de récupérer le contenu d'une URL est simplement de le demander.

```
'http://stfx.eu/small.html' asUrl retrieveContents.
```

Flots (Streams) dédiés

Dans un vrai langage typé dynamiquement, le type (la classe) d'un objet n'est pas l'aspect le plus important. Ce sont plutôt les messages compris par cet objet qui importe le plus.
Il existe de nombreuses entrées et sorties basées sur des streams dans Zinc HTTP components. Le code en charge de ces IO se cantonne à un ensemble réduit de messages que ces streams se doivent de comprendre. On peut remplacer un stream par un autre, même lorsqu'il ne partage pas la même superclasse aussi longtemps qu'ils implémentent (comprennent) les mêmes messages.
Prenons l'exemple de l'encodage d'un transfert HTTP en tronçon. Il s'agit d'une technique optionnelle dans laquelle le serveur décide de ne pas retourner un document en un seul morceau, mais au contraire de le découper en plusieurs tronçons, chacun précédé par sa taille jusqu'à ce qu'un dernier tronçon vide termine le stream. Cette technique est la plus fréquemment utilisée pour transmettre des fichiers générés dynamiquement et dont la taille n'est pas connue à l'avance. Lorsqu'un client lit une réponse, un transfert de ce type est signalé dans l'en-tête (header) HTTP. Un objet auxiliaire (helper object) appelé le lecteur d'entité examine les en-têtes, crée le stream adéquate et demande à l'objet réponse de se lire lui-même.
Sans que le code readFrom: de la méthode ne le sache, le stream est alors modifié. Dans le cas d'un transfert en tronçons, le stream d'origine est enveloppé dans un objet ChunkedReadStream qui gère le traitement des tronçons de façon transparente.
Les messages next ou atEnd fonctionnent comme avant, mais ChunkedReadStream récupère les tronçons de la taille indi-

quée dans le stream et les bufferise autant que nécessaire.

```
ZnChunkedReadStream>>#next
 self ensureChunkOrAtEnd.
 self atEnd ifTrue: [ ^ nil ].
 ^ chunk at: (position := position + 1)
```

Sur le même modèle, il existe des streams qui ajoutent du buffering à des streams, décodent des octets en caractères selon un encodage spécifique, convertissent les terminaisons de ligne, etc.
Ce mécanisme d'enveloppe peut s'appliquer plusieurs fois: un stream peut décoder des octets en caractères en provenance d'un stream qui décode du GZIP qui lui-même provient d'un stream recomposé à partir d'une réponse en tronçons. Ainsi un bloc de données passe de encrypté à décrypté, puis réassemblé à partir de tronçons et finalement décodé selon un jeu de caractères.
Tout ceci se produit de façon transparente dans Zinc et sans avoir besoin de partager une superclasse commune, juste en implémentant le même jeu de messages dans chaque classe de stream.
L'avantage est que chaque fonctionnalité (l'interprétation du transfert fragmenté, la décompression GZIP, le décodage de caractères, la mise en mémoire tampon) est encapsulée dans un seul objet et que vous pouvez les combiner selon vos besoins.

Objets de support

Une façon de démarrer une nouvelle conception d'objet consiste à créer un langage spécifique à un domaine (DSL) à l'aide d'objets de support. Ces objets sont souvent plus faciles à définir et à mettre en œuvre. En encapsulant des particularités, ils sont néanmoins très utiles. Dans le domaine des protocoles HTTP et Internet, il existe de nombreux objets de ce type.
Être capable d'analyser et d'interpréter les parties constitutives d'une URL, générer une URL correctement encodée construite à partir de parties, s'intègre parfaitement avec un seul objet. Des APIs adaptées facilitent le travail avec les objets URL. Un comportement important comme la construction d'une nouvelle URL à partir d'une URL relative dans le contexte d'une URL existante trouve ici sa place naturelle.
Une fois analysés les objets URL connaissent tous les détails. C'est par exemple le cas avec le message asUrl envoyé à une chaîne,

```
'http://www.google.com/search?q=Smalltalk' asUrl queryAt: #q.
 "=> 'Smalltalk'"

'http://www.google.com/search?q=42' asUrl authorityWithPort.
 "=> 'www.google.com:80'"

'http://www.google.com/search?q=42' asUrl firstPathSegment.
 "=> 'search'"
```

The withRelativeReference: message implements a pretty tricky algorithm.

```
'http://www.site.com/static/html/home.html' asZnUrl with
RelativeReference: '../js/menu.js'.
 "=> http://www.site.com/static/js/menu.js"
```

Les deux expressions suivantes génèrent la même URL, https://encrypted.google.com/search?q=Smalltalk%20en%20Fran%C3%A7ais. Notez l'encodage. Tout d'abord, une API de style constructeur (builder) est utilisée, puis des messages binaires de commodité sont utilisés.

```
ZnUrl new
 scheme: #https;
 host: 'encrypted.google.com';
 addPathSegment: 'search';
 queryAt: #q put: 'Smalltalk en Français';
 Yourself.

'https://encrypted.google.com' asUrl / #search ? (#q ->
'Smalltalk en Français').
```

MimeType, Cookie ou CookieJar sont développés suivant la même approche.
Une autre catégorie d'objets de support implémente des algorithmes. Les exemples sont CharacterEncoder, BasicAuthenticator, PercentEncoder ou Base64Encoder. Placer uniquement le code d'un algorithme dans une seule classe peut avoir beaucoup de sens en raison du principe de responsabilité unique. Cela n'empêche pas d'ajouter un certain nombre de méthodes pratiques ici et là pour mieux s'intégrer aux classes standards.
Par exemple, encoder en Base64 un ByteArray avec 10 valeurs de values 0 à 9 produit la String 'AAECAwQFBgcICQ=='.

```
ZnBase64Encoder new encode: #[ 0 1 2 3 4 5 6 7 8 9 ].
#[ 0 1 2 3 4 5 6 7 8 9 ] base64Encoded.

ZnBase64Encoder new decode: 'AAECAwQFBgcICQ==.
'AAECAwQFBgcICQ==' base64Decoded.
```

Un message de commodité unaire fait le travail dans la plupart des cas. Cependant, l'objet de support actuel a plus d'options: changer l'alphabet utilisé, si et comment couper les lignes, quel caractère de nouvelle ligne utiliser, comment faire le remplissage, quel caractère de remplissage utiliser, à quel point l'analyse doit être stricte. Ces options ne peuvent pas toutes être exprimées sous forme d'arguments sans augmenter à nouveau la complexité. Mais c'est le prix à payer.

Journalisation (Log) des objects

Pendant le développement ainsi que pendant la production, la journalisation et les métriques sont un outil important. Dans la plupart des cas, cela se fait en écrivant des messages textuels dans les flux de journalisation. L'interprétation de ces flux est souvent très difficile. La réimplémentation des frameworks de journalisation ne résout pas les problèmes fondamentaux.
La journalisation d'objets est une solution élégante, puissante et efficace. L'idée est simple: nous instrumentons notre code pour qu'il génère des objets log réels lors de son exécution. Pharo offre la possibilité d'ajouter un journal des objets au lieu de simplement du texte. Ce flux d'objets de journal peut ensuite être utilisé à des fins d'inspection pendant le développement ainsi que pour la journalisation textuelle classique, la surveillance du système et les calculs de métriques.

Le design est tout aussi simple. Nous créons une hiérarchie d'objets de journal personnalisés sous la superclasse commune LogEvent, elle-même une sous-classe d'Annoncement. Pour distribuer des objets d'événement, nous utilisons un annonceur. Cette classe système délivre des annonces à ses abonnés, le cas échéant, éventuellement filtrées par type.
Dans Pharo, vous pouvez consulter l'expression suivante pour commencer.

```
ZnLogEvent announcer.
```

Dans l'un des onglets de l'inspecteur, vous verrez des LogEvents arriver tandis que Zinc HTTP components exécute le code côté client ou serveur. **Figure 2**
Les LogEvent instances sont créées, remplis de données utiles, puis émis comme suit :

```
ZnLogEvent>>#emit
 self announcer announce: self
```

Puisque nous créons déjà de nombreux objets pendant l'exécution de toute façon, comme des requêtes, des réponses, des URL, etc., qui ne sont normalement pas utilisés par la suite, les emballer dans un événement est bon marché. Ce qu'il advient de ces événements est décidé ultérieurement par le code qui y souscrit.
En implémentant des méthodes printOn: appropriées sur chaque événement, nous pouvons établir une manière régulière de les imprimer, couvrant le cas classique de la journalisation textuelle. L'expression suivante configure un écouteur pour nos événements de journal et les imprime dans la fenêtre de transcription.

```
ZnLogEvent logToTranscript.
```

L'exécution d'une simple requête HTTP GET entraînera la sortie suivante.

```
ZnClient new beOneShot get: 'http://stfx.eu/small.html'.

2020-08-14 14:04:00 022 Connection Established stfx.eu:
80 146.185.177.20 121ms
2020-08-14 14:04:00 023 Request Written a ZnRequest(GET
/small.html) 0ms
2020-08-14 14:04:00 024 Response Read a ZnResponse(200 OK
text/html 113B) 29ms
2020-08-14 14:04:00 025 GET /small.html 200 113B 29ms
2020-08-14 14:04:00 026 Connection Closed 146.185.177.20:80
```

Au niveau de la journalisation standard du client HTTP, 5 événements de journal ont été générés : un ConnectionEstablishedEvent, un RequestWrittenEvent, un ResponseReadEvent, un ClientTransactionEvent et enfin un ClientConnectionClosedEvent. Comme vous pouvez le voir, les informations de synchronisation sont également incluses.
Pour surveiller le débit sur un serveur, nous pourrions nous abonner à ServerTransactionEvents et les compter. Puisque nous avons également accès à la requête et à la réponse de chaque transaction, nous pouvons calculer la quantité d'octets transférés. Avec l'accès à l'URL et à tous les en-têtes, nous pourrions ajouter toutes sortes de filtres supplémentaires.
La génération d'une sortie au standard industriel Apache Common Log Format se réduit simplement à utiliser le formateur approprié.

```
| formatter |
formatter := ZnCommonLogFormat new.
ZnLogEvent announcer
 when: ZnServerTransactionEvent
 do: [ :event |
  formatter format: event on: Transcript.
  Transcript cr; endEntry ].
```

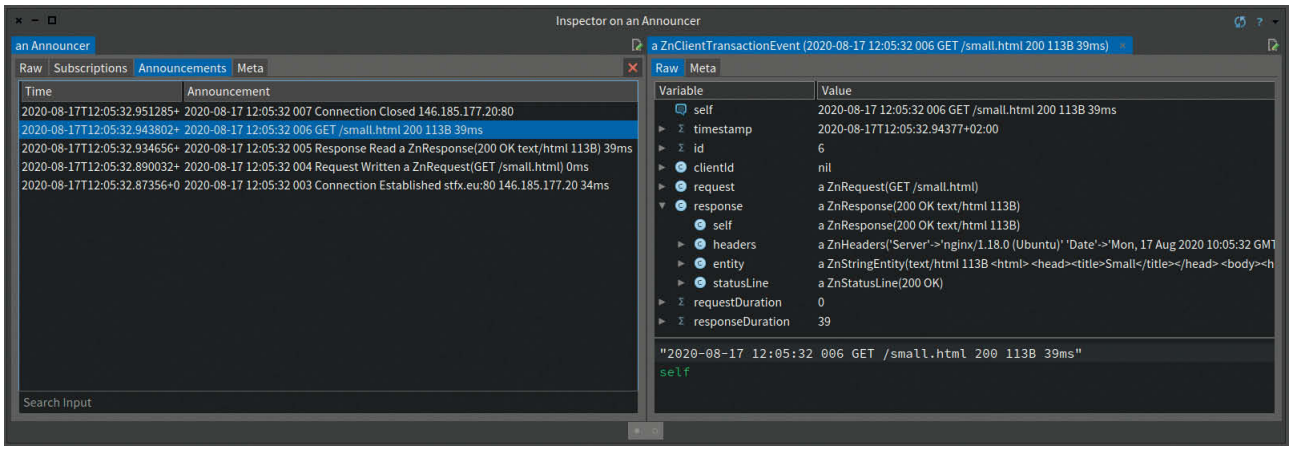
Ici, le formateur trouve toutes les données dont il a besoin dans la requête et la réponse intégrées de l'événement.

Conclusion

La simplicité du langage et du modèle d'objet de Smalltalk (des messages à la syntaxe minimale et un modèle d'héritage simple) donne in fine un outil étonnamment puissant pour exploiter pleinement la conception d'objets dans sa forme la plus pure.
J'espère que cet article a suffisamment suscité votre intérêt pour télécharger Pharo Smalltalk et commencer à explorer par vous-même. Il y a tellement à apprendre.

Zinc: https://github.com/svenvc/zinc

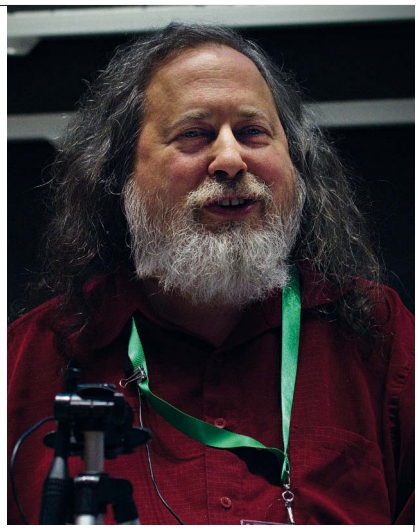
Figure 2



La banque d'Angleterre ouvre la voie aux « britcoins »

La Grande-Bretagne rejoint la liste des pays envisageant une cryptomonnaie souveraine. En effet, la banque d'Angleterre et le ministère des Finances du pays ont annoncé la création d'un groupe de travail chargé de se pencher sur cette idée. Le projet envisage la mise en place d'une monnaie numérique de la banque centrale, à l'instar du Yuan numérique mis en œuvre par la Chine ou le e-krona expérimenté en Suède.

Le retour en fanfare de Richard Stallman



© Ruben Rodriguez

Le saint patron du logiciel libre Richard Stallman a fait son retour au conseil d'administration de la Free Software Foundation, qu'il avait quitté en 2019 suite à des prises de position un peu hasardeuse dans le cadre de l'affaire Epstein. Un retour sans tambour ni trompette qui a été accueilli plutôt froidement par le petit monde du libre : l'annonce a ainsi provoqué la démission du président de la fondation et de plusieurs membres du conseil d'administration, mais aussi la fin du soutien financier de Red Hat et Fedora et les critiques des communautés SUSE, Debian et de The Document Foundation. Mais le père fondateur du projet GNU ne semble pas s'en émouvoir outre mesure.

FloC, le renouveau du tracking selon Google

Le cookie de pistage publicitaire a du plomb dans l'aile et Google commence à réfléchir aux alternatives. Le géant de la publicité en ligne a d'ailleurs déjà commencé à déployer à titre expérimental sa solution de rechange : FloC, acronyme de Federated Learning of Cohorts. Pour expliquer rapidement le principe : plutôt que de suivre individuellement le comportement des individus, Google préfère les agglomérer dans des « cohortes » et se baser sur ces résultats agrégés pour diffuser de la publicité ciblée. Mais la solution ne récolte pas vraiment l'approbation de l'écosystème : l'EFF a qualifié FloC de « très mauvaise solution », poussant les navigateurs Brave et Vivaldi à annoncer qu'ils ne supporteraient pas la technologie. Edge, Safari et Firefox ont préféré opter pour une approche « Wait and see » tandis que chez Wordpress, on réfléchit à désactiver par défaut le support de cette nouvelle fonctionnalité sur les sites web utilisant son CMS. Autant dire que FloC doit encore faire ses preuves.

Exchange : le FBI s'occupe de la désinfection

Les failles affectant Microsoft Exchange dévoilées début mars ont conduit les agents du FBI à prendre des mesures drastiques. Ceux-ci ont annoncé avoir obtenu l'autorisation du ministère américain de la Justice de pouvoir se connecter aux serveurs américains compromis par des attaquants et supprimer les portes dérobées qui y avaient été installées. Une opération qui s'est déroulée en toute discrétion, le FBI ayant choisi de ne pas avertir les administrateurs des serveurs concernés en avance afin de ne pas ébruiter l'affaire. Les autorités fédérales s'affairent maintenant à signaler aux administrateurs qu'ils sont passés faire un tour sur leurs serveurs.

Poutine en veut-il au CNED ?

Au premier jour du passage au travail à distance pour les élèves et étudiants français, les serveurs ont eu bien du mal à tenir la charge, laissant les étudiants et leurs professeurs en rade plusieurs jours. Le ministre de l'Éducation Jean-Michel Blanquer s'est récolté une avalanche de commentaires, parfois cinglants, en évoquant « des cyberattaques venues de l'étranger » qui auraient notamment visé le service « Ma Classe à la maison » du CNED. Une affirmation pourtant confirmée par les enquêteurs, qui évoquent « des attaques provenant de Chine et de Russie ». Mais difficile d'en tirer des conclusions : les commanditaires de l'attaque ont en effet pu se dissimuler derrière des serveurs piratés dans ces pays pour brouiller les pistes. L'attribution n'est pas une science exacte, loin de là.

Signal cherche la faille chez Cellebrite

Dans un post de blog cinglant, le créateur de Signal s'est amusé à décortiquer plusieurs équipements commercialisés par la société Cellebrite, société spécialisée dans la revente d'outils d'analyse des téléphones portables à destination des forces de l'ordre. Il a mis à jour plusieurs failles permettant à une application installée sur un téléphone analysé par Cellebrite de fausser les rapports produits, voire de saboter les rapports futurs de l'outil. Il se dit prêt à en dévoiler les détails si Cellebrite accepte de révéler aux fournisseurs les failles que la société exploite pour contourner les protections mises en œuvre par les fabricants de smartphones.

INFORMER
pour transformer l'entreprise

La dématérialisation, le Cloud,
les communications unifiées,
les nécessités de la cybersécurité
transforment le travail et toute l'entreprise,
les services publics.
Le magazine, le site, ses newsletters
vous informent sur cette actualité mouvante
et vous aident à décoder les tendances

Lire le magazine en digital

www.solutions-numeriques.com/magazines/solutions-numeriques-n31/

NOUVEAU : TELETRAVAIL, comparatifs de MATERIEL

- Vous êtes responsable informatique ou bien dirigeant ou cadre d'entreprise ?
  2 sites répondent à votre profil
- La cybersécurité vous concerne ?
  Cliquez sur l'onglet. Vous trouverez les infos, l'annuaire, le lexique, etc
- L'emploi, les salaires, les formations, les offres vous intéressent ?
  Le site sur l'Emploi dans le numérique est à votre disposition.

www.solutions-numeriques.com

François Tonic

Emploi & salaire : au-delà du Covid !

Depuis un an, nous alternons entre restrictions et restrictions. Le télétravail imposé ne favorise ni les échanges ni la cohésion des équipes. Mais au-delà de toutes ces difficultés, on constate tout de même un marché de l'emploi qui cherche des solutions, car les entreprises, les ESN ont toujours besoins de développeurs et des profils techniques. Et les problèmes de recrutement demeurent même si aujourd'hui, ils paraissent moindre.

La longue crise du Covid n'a pas annulé les projets IT, ni les développements, du moins, pas en masse. Les reports ont été nombreux, mais à un moment donné, il faut relancer les développements. Et les entreprises sont nombreuses à chercher des développeurs, des techniciens, des responsables réseau, etc. Il y a un an, nous avions mis en évidence le contrecoup violent parté à de nombreux freelances.

Ainsi une étude Michael Page / Chooseyour boss repositionne certains chiffres intéressants sur un marché très inégal :

- **79% des professionnels de l'IT (toutes expertises et toutes formations confondues) considèrent que le marché est moins favorable depuis le début de la crise sanitaire**.
- **36% en recherche de poste n'ont aujourd'hui aucune piste d'emploi tandis que 42% déclarent en avoir une ou deux**
- 52 % pensent recherche un emploi dans les 3 mois
- 78 % des entreprises de l'étude pensent recruter en 2021

Les freins restent souvent identiques à la situation ante-Covid : comment attirer les talents et les bonnes compétences, tension sur certains profils, difficulté de trouver selon la ville / région.

Côté talent.io, une récente étude mettait en avant :

- **60% de propositions d'entretien par candidat à Paris** comparé aux 6 mois pré COVID-19
- **Les profils juniors sont les plus affectés par la pandémie**. Les développeurs et les développeuses juniors ont enregistré une baisse de 3% de leur salaire (médian) alors que les salaires proposés aux candidats de plus de 4 ans d'expérience sont restés stables
- **Les développeurs et développeuses travaillant en Régions sont confrontés à une plus forte baisse de leurs salaires**, jusqu'à -5% sur le segment des candidatas avec 4 à 6 ans d'expérience. Pourtant, la demande de candidats expérimentés est en croissance : le volume de demandes d'entretien est en hausse de 13% en moyenne.
- **Globalement les salaires médians sont 15% plus élevés à Paris**

Les profils les plus demandés selon Talent.io sont : développeurs full stack, Node, React, PHP. Kotlin, TypeScript et Redux sont les compétences en pleine croissance. Et Paris reste la plus active avec le double de proposition qu'en province. Ce dernier est toujours aléatoire selon le profil. Cette étude montre que le full stack est en tête de demande dans toutes les métropoles. Le développeur front end est le moins recherché.

Que serait une étude sur l'emploi sans les salaires ? Talent.io estime que le salaire moyen à Paris est de 46 000 € contre 40 000 en régions. Attention : ce salaire moyen prend entre les développeurs, les architectes logiciels et les dévs leads. Ces deux derniers profils augmentent de facto le salaire moyen, car ce sont des postes généralement à plus salaire élevé. (**voir tableau 1**)

Il est difficile de se faire une idée précise quand les études utilisent uniquement des termes génériques : dév mobile, dév full stack, dév front end, dév back end. Pour nous, être Lead Dev sans expérience préalable, est une non-sens. Pour le reste, peu de surprises :

- 34 – 38 k€ en région pour un profil développeur, contre 38-45k€ sur Paris pour 0 à 3 ans d'expérience
- env. 50 k€ pour une expérience de 4-6 ans sur Paris, 40-42 k€ en région

Comme toujours, il faut interpréter avec prudence ces tendances. La pression Covid est incontestable, nous le voyons dans d'autres études. La crise actuelle impacte réellement les salaires : soit stagnation d'une année sur l'autre, soit une baisse. Pour les profils juniors, la baisse n'est pas forcément

Un chômage à surveiller

Le chômage des informaticiens (terme générique) reste une réalité, notamment dans les régions fortement touchées par la crise du Covid et dépendantes de certaines industries (aéronautique, automobile). Ces régions peuvent voir le chômage en hausse depuis 1 an.

Les mesures de chômage partiel ont amorti l'impact sur l'emploi, mais le chômage dans les métiers de l'informatique a toujours existé. Il faudra sans doute attendre quelques mois pour voir la situation réelle, notamment pour les profils 45-55ans.

Tableau 1

	Paris			Province		
Expérience (en années)	Ante-mars 2020	Post-mars 2020	Tendance	Ante-mars 2020	Post-mars 2020	Tendance
0-2	39	38	-	33	34	+
2-3	45	44	-	37,5	36,5	-
4-6	50	50	=	42	40	-
+7	55	55	=	46	45	-

En k€. Salaires bruts.

UN RECRUTEMENT TOUJOURS ACTIF

La liste est donnée à titre informative. Elle n'est pas exhaustive et le recrutement peut être clos à la publication de ce dossier. Mais cela montre le relatif dynamisme du marché.

Société	Nombre de postes ouverts
Ippon	150
Akeneo	+100
Avisto	80
Blue Soft	100
Sopra Steria	2500
Devoteam	1200
Antemeta	55
Itancia	85
Neoxia	40
Umanis	600
Betoobe	20

Nous avons privilégié les recrutements de profils techniques / développeurs, mais les campagnes de recrutement sont parfois très larges. Ces recrutements sont dans toute la France.

une surprise : ces profils ont été les premiers impactés au printemps 2020 puis par l'arrivée de nombreux nouveaux arrivants. CodinGame, en février dernier, proposait des salaires moyens suivants :

- Dév mobile : 35 300 €
- Dév jeux vidéo : 37 000 €
- Dév full stack : 37 000

Cette étude a été faite au niveau mondial et non spécifiquement en France.

Kicklox fournit une autre vue intéressante :

	Junior	Senior
Développeur React	38000€	62000€
Développeur Angular	36500€	58000€
Développeur Vue	36500€	58000€
Développeur jQuery	31000€	47000€
Développeur Android / iOS natif	36000€	62000€
Intégrateur HTML/CSS	26000€	45000€

Nous n'avons jamais été fan des segmentations : développeur React, Angular, etc. Car cette segmentation n'existera plus dans 2-3 ans, d'autres technologies arriveront. Pour nous, il est préférable de parler de développeur JavaScript, terme générique qui couvre l'ensemble des technologies liées.

Sur la partie Java, Ruby, C#, etc., les fourchettes sont, comme souvent, larges : 28 à 33 k€ pour un junior, 54-65 k€ pour un senior. Mais selon le langage, les écarts sont énormes. Pour un dév PHP, on tournera vers 54 k€ contre 65 pour un dév Java. Bien entendu, cela va dépendre du type de société, la ville, etc. Par contre, pour un profil dit full stack (terme flou et fourre-tout), on débutera, selon Kicklox, vers 33 k€, ce qui est finalement une fourchette (très) basse. Pour un full stack, les exigences techniques sont élevées et cela doit se traduire par le salaire proposé.

Comme toujours, la formation reste un élément important :

- Se former ou s'auto-former
- Passer les certifications
- Ne pas hésiter à sortir de sa sphère technique

D'ailleurs, des ESN profitent du Covid pour former les développeurs.

Les métiers du développement, de l'intégration et des tests en France

Le réseau Grande Ecole du Numérique publie les tendances des métiers du numérique dans toute la France. Ces données datent de l'automne 2020, certaines données datent de 2019. Nous prenons en compte uniquement les % des profils développement / intégration / testeur. Ce chiffre est le poids en % dans les métiers du numérique dans ladite région.

Région	Emplois développement/ intégration/tests (en %)	Le top 3 des profils techniques
Auvergne-Rhône-Alpes	39 %	Fullstack Front-end DevOps
Bourgogne Franche-Comté	30 %	Télécom Fullstack Front-end
Bretagne	32 %	Fullstack DevOps Front-end
Centre Val de Loire	19 %	Télécom Full-stack Développeur business
Corse	10 %	Télécom
Grand Est	23 %	Télécom Chef de projet fullstack
Hauts de France	37 %	fullstack DevOps Back-end
Île de France	35 %	Fullstack Développeur business Front-end
Normandie	16 %	Télécom Community manager fullstack
Nouvelle Aquitaine	35 %	Fullstack DevOps Back-end
Occitanie	32 %	Fullstack DevOps Chef de projet web
Pays de la Loire	48 %	Fullstack Front-endBack-end
Provence Alpes Côte d'Azur	36 %	Fullstack Front-end DevOps

On constate que les profils sont plutôt homogènes dans l'ensemble des régions. Certaines sont demandeuses de compétences en télécom : maintenance, installation.

Au final ?

Difficile de savoir où ira réellement 2021. Sauf reprise très forte dans les prochains mois, nous serons sur un marché identique à 2020 sans évolution importante des salaires. Il faut que les nouveaux arrivants comprennent la situation, ce qui n'est pas toujours le cas. Un certain réalisme s'est installé depuis 1 an. Les nouveaux développeurs arrivent sur un marché difficile avec une concurrence accrue.

Mauvais recrutement ?

Le risque de recruter le mauvais profil n'est jamais écarté. Une récente étude du cabinet Robjert Half montre une partie des effets que cela peut avoir. Il faut presque 4 mois pour se séparer d'un mauvais profil entre prendre acte de la situation et le départ de la personne. L'étude met en avant que les trois quarts des responsables ont déjà recruté le mauvais profil et que cela a un impact négatif. Et cela peut impacter l'équipe, être une perte de temps pour tout le monde.

Pourquoi un mauvais recrutement ? Selon l'étude, la précipitation dans le recrutement est un facteur important : des étapes de recrutements écourtées, l'absence de rencontre physique, des évaluations plus difficiles à réaliser.

Une étude datant de 2018 disait qu'il fallait 2 semaines pour savoir si la nouvelle recrue correspondait ou non aux attentes.

Coralie Nohel

Directrice du Recrutement chez Zenika, je suis passionnée par les nouvelles approches et expertises de mon domaine qui remettent la balle au centre entre recruteurs et candidats. Dans ma manière d'exercer mon métier, tout s'articule autour d'une conviction : soigne ton candidat il te le rendra bien, et traite avec des personnes et non des dossiers !

Les enjeux du recrutement dans les métiers de la tech : à l'ère du covid-19

AVEC LA PARTICIPATION DE :

Alexis

Après un doctorat dans le domaine des sciences des matériaux à l'EPFL, j'ai effectué une transition professionnelle vers le monde du numérique. Actuellement en mission chez la MAIF, je suis déterminé à transformer un concept en une réalité du marché.

Jordan

Curieux de tout et créatif dans l'âme, l'informatique me permet de m'immerger dans des domaines complexes, en proposant des solutions innovantes, et en prenant en compte les contraintes des environnements dans lesquels j'évolue. Saupoudrez d'une dynamique d'équipe bien huilée, et vous créez l'environnement idéal à mon épanouissement.

Alison

Ancienne développeuse logiciel, j'aide désormais les équipes et/ou les projets à mieux s'organiser et communiquer pour développer le bon produit et à le faire bien. Grâce à des pratiques agiles et un œil extérieur sur les problèmes de mes clients, je pose les bonnes questions pour que les solutions émergent du collectif.

Julien

Développeur depuis près de 15 ans, j'aime me confronter à de nouveaux défis.

Quels ont été les impacts du 1er confinement sur le marché de l'IT ?

Alison, consultante Zenika à Grenoble : "Décembre 2019, j'ai quitté mon job pour profiter d'une année de voyage en tour du monde. Dans ce genre de cas, tu dis au revoir à tous tes potes, ta famille, tu vends tes biens pour partir l'esprit léger et à aucun moment, tu ne t'imagines de retour au bout d'à peine deux mois et demi… Et pourtant, mon avion retour a bien atterri à la mi-mars sur le sol français et je suis allée m'installer provisoirement dans la région Centre. Et voilà qu'il fallait reprendre le chemin du travail avec ce contexte de pandémie mondiale, en plein confinement."

Il est incontestable que la crise COVID-19 a complètement bouleversé les entreprises, les secteurs d'activités et les économies sur son passage. Mais, et c'est ce qu'illustrent parfaitement les propos d'Alison en introduction, cette crise a également impacté nos vies et a réussi en très peu de temps à faire évoluer les mentalités dans nos quotidiens professionnels.

Le marché du travail dans son ensemble a dû s'adapter à une vitesse fulgurante pour faire face à cette situation ; les chefs d'entreprises, les services RH, les salariés, les chercheurs d'emploi, tous ont été pris de court et ont dû parfois se réinventer.

"Cette situation où tout était au ralenti a eu beaucoup de bon pour ma carrière. J'avais enfin du temps pour réfléchir à ce que je voulais faire de ces 8h par jour. Mon dernier poste, c'était développeuse logiciel et j'avais pris un rôle de Scrum Master à mi-temps sur mon équipe. Et c'est là où je me suis le plus éclatée. Pour réorienter ma carrière vers des postes avec une plus forte composante agile, j'ai passé une bonne partie de la période de confinement à consommer beaucoup de contenu sur des thématiques d'agilité. J'ai commencé à prendre part aux communautés d'agilistes en ligne et j'ai décidé de passer des certifications pour être plus facilement employable sur des métiers de Scrum Master / d'accompagnement d'équipes agiles. Et puisque j'avais le temps, j'ai choisi de contribuer moi aussi à la diffusion de ces méthodes de travail en créant ma chaîne YouTube sur le thème de l'agilité.

Pour faire court, grâce à ce contexte particulier, j'ai pu prendre le temps de réfléchir à la suite de ma carrière et de me focaliser sur ma montée en compétences" poursuit Alison.

Le marché du recrutement IT n'a pas été épargné par cette crise

D'après les chiffres du Syntec, une baisse de CA de 4,6 % a été enregistrée en 2020 dans les secteurs du numérique. La situation est contrastée selon les métiers. Les éditeurs de logiciels s'en sortent mieux, les ESN ont fait partie des structures assez fortement impactées avec une baisse de CA dans le secteur de 4,2 %. Les activités de conseil en technologies sont plus particulièrement touchées avec un repli de -12,3 % en raison de leur exposition aux secteurs aéronautiques et automobiles durement touchés par la crise Covid-19.

Les épisodes de confinement, et principalement le premier en mars, ont eu un impact direct sur les offres d'emploi. L'Apec annonce pour 2020 une baisse de 29 % du volume de recrutements cadres tous secteurs confondus. En avril 2020 notamment, les offres d'emploi cadres publiées sur apec.fr ont chuté de 62 % par rapport au même mois en 2019. Indeed, agrégateur d'offres d'emploi important du marché de l'emploi, a également vu son flux d'offres d'emploi baisser de 31 %. Le nombre d'offres destinées à des emplois pour jeunes et débutants a particulièrement été impacté (-51 % des offres pour cette population).

Ressources :
https://syntec-numerique.fr/actu-informatique/bilan-2020-perspectives-2021-secteur-numerique-secteur-ne-ralentit-pas-autant-que
https://corporate.apec.fr/home/nos-etudes/toutes-nos-etudes/secteur-des-activites-informatiq.html

Quels ont été les impacts sur le recrutement en particulier pour une entreprise comme Zenika ?

Comme dans tout secteur de prestation de services, l'activité de Zenika s'est trouvée limitée et conditionnée à la situation économique des clients. L'épisode 1 a fortement impacté l'activité de recrutement. Dès début avril, nous avons décidé de ralentir très fortement les embauches initialement prévues au plan de recrutement. Naturellement, l'urgence était tout d'abord de gérer les missions qui étaient stoppées par nos clients et de gérer les intercontrats de nos consultants en CDI. À fin mars 2020, 30 embauches avaient déjà été faites, et plusieurs étaient déjà conclues pour les mois menant à l'été. Rapidement et malgré ce contexte flou, nous avons décidé de maintenir les contrats de nos nouveaux embauchés. Et ce, qu'ils soient en mission ou non et surtout, même s'ils étaient encore en période d'essai. Pour nous, le recrutement est un véritable investissement. Nous y passons du temps et y mettons beaucoup d'énergie. Il était donc primordial de faire le maximum pour préserver ces talents qui nous ont fait confiance, en conservant et valorisant leurs compétences en interne.

Deuxième décision importante, nous n'avons pas remis en cause les embauches planifiées dans les mois qui venaient, malgré l'incertitude économique que nous vivions et la quasi-certitude qu'il serait difficile de leur trouver des missions pendant le confinement, pour les mêmes raisons citées précédemment.

Enfin, nous avons décidé de ne pas geler complètement nos embauches, nous sommes passés à une approche de "slow recruitment", nous avons revu et resserré notre cible d'embauche sur des profils plus opérationnels et expérimentés. C'est un effet "plug and play" qu'ont dû adopter beaucoup d'entreprises, et c'est un syndrome connu en périodes de crises. On se recentre sur l'essentiel, on recherche la valeur sûre, et on réduit la prise de risque.

Jordan, consultant Zenika à Lyon "J'étais freelance jusqu'en août 2020, entre les deux confinements. Je me suis mis en recherche d'un projet qui me ferait vibrer, et d'une équipe que j'aurais envie d'accompagner. C'est à ce moment que j'ai rencontré mes principaux obstacles :

- *des missions très ennuyeuses, en tout cas qui n'étaient pas à mon goût*
- *des recruteurs et des clients qui insistent pour recruter des profils très spécialisés, ou au CV impressionnant.*

J'avais en plus de ça souvent à faire à des recruteurs qui n'avaient que peu d'intérêt pour mon profil, au profit de la possibilité qu'ils voyaient de placer quelqu'un chez leur client. Mais, apparemment mon profil ne plaisait pas. C'était souvent : trop jeune, trop généraliste, trop cher. J'ai globalement rencontré une grande fermeture d'esprit tant de la part des recruteurs, que des commerciaux et des clients, probablement à cause d'une gestion du risque un peu zélée. Même si à aucun moment je ne me suis dit que je ne trouverais pas du travail à cause de la situation sanitaire, j'en étais arrivé à me dire que je ne trouverais pas de missions qui m'intéressaient, et qu'il me faudrait me former à des technologies plus conventionnelles, ou plus demandées."

Avant la crise, l'emploi dans la filière informatique était au beau fixe, on parlait même de secteur en tension : beaucoup d'offres de la part des entreprises et une pénurie de profils spécialisés en face. Les entreprises n'ont eu d'autre choix que d'ouvrir largement leurs postes à d'autres types de profils, juniors, candidats en reconversion professionnelle, stagiaires et apprentis ainsi qu'autodidactes. Mais suite à la crise, ce sont ces profils qui ont les premiers pâtis de ce resserrement du marché.

"Malgré ma volonté à trouver un job dans le domaine de l'Agilité, je me suis confrontée à une réalité de marché où les entreprises attendaient des candidats déjà séniors. J'ai ressenti que ce n'était pas le moment de prendre des risques avec des profils juniors qu'on doit prendre le temps de faire monter en compétences. Et j'avoue que j'ai plusieurs fois remis en question le choix que j'avais fait de réorienter ma carrière." complète Alison.

Zenika accompagne chaque année des stagiaires de fin d'études, avec une embauche à la clé, si le stage se passe bien des deux côtés. Habituellement, les stages sont presque systématiquement transformés en CDI, et notre capacité à trouver des missions à ces profils juniors ne nous inquiète pas. De plus, cela fait partie de notre ADN de transmettre et de faire grandir en accompagnant des profils à potentiel vers l'expertise. Ce fut un peu différent en 2020. Nous avons pourtant tout mis en œuvre pour honorer notre promesse, même en cette période. À Lille, l'annonce du confinement ne nous a pas freinés pour confirmer l'un de nos stagiaires en CDI en avril. À Nantes, Alexis a finalement pu nous rejoindre, même si nous avons exceptionnellement dû conditionner sa confirmation en CDI au fait que nous lui trouvions une mission.

"Suite à une reconversion professionnelle dans le domaine de l'informatique, j'ai effectué un cursus de master qui a culminé en un stage de fin d'études chez Zenika débutant en janvier 2020. J'ai obtenu ce dernier à la suite d'un processus de recrutement identique à celui menant à l'embauche d'un consultant en CDI. Zenika a en effet pour volonté de faire évoluer ses stagiaires vers un poste de consultant dès la fin du stage. Le processus de recrutement a donc eu lieu en présentiel durant l'été 2019. Le déroulement du stage a été légèrement impacté par la mise en place du premier confinement en mars 2020, mais cela n'a pas empêché la bonne poursuite du projet, et a même permis la mise en place d'un second sujet.

Au vu de la situation, et avant même la fin de mon stage, fixée à mi-juillet, il est cependant apparu évident que mon embauche en CDI, initialement prévue pour début septembre, allait être remise en question. Les premières interrogations se sont posées dès la fin avril, et Antoine Bonneau, le directeur de l'agence de Nantes m'a même directement appelé afin de faire part de son embarras quant à la situation dès la fin Mai. Il m'a cependant expliqué que tout était mis en œuvre afin de trouver une mission me permettant de débuter mon CDI à la date prévue. De mon côté, je lui ai réaffirmé ma volonté de

continuer à travailler chez Zenika tout en lui indiquant que j'étais prêt à décaler mon début de CDI de quelques mois si cela me permettait de rester.
Durant l'été 2020, les perspectives de missions étaient très faibles du fait du contexte et de la période de congés. De fait, j'ai été amené à explorer d'autres pistes tout en restant en contact avec Zenika. J'ai donc effectué quelques entretiens en présentiel avec d'autres entreprises spécialisées dans le conseil. Mi-août, j'ai été recontacté par Zenika pour une potentielle mission. J'ai donc été mis en relation avec le client, et les processus d'entretien et de validation technique se sont déroulés entièrement à distance. Je n'ai pas ressenti de limitation liée à ce format, et j'ai au contraire trouvé le contact moins formel, plus direct et plus agréable dans son ensemble. De plus, cela m'a permis d'effectuer l'entretien technique via un partage d'écran, ce qui m'a semblé moins contraignant et moins stressant que d'avoir plusieurs personnes physiquement en train de me regarder résoudre le problème posé. J'ai été retenu sur la mission par le client et ai pu rejoindre Zenika en CDI comme prévu à la signature de mon stage" Alexis, consultant Zenika à Nantes.

Comment s'adapter dans un contexte en 2020 peu favorable au changement d'entreprise, et des interactions devenues quasiment 100 % à distance ?

Pour les entreprises qui n'avaient pas gelé complètement leurs recrutements, les recruteurs se sont heurtés à de nouvelles difficultés. Chercher un nouveau travail, ce n'est pas forcément évident en temps normal, alors pendant une période de pandémie anxiogène, entrecoupée de confinements, de distanciations sociales et de complications économiques pour les entreprises, c'est encore plus ardu pour les candidats de se projeter à changer d'entreprise. Sans grande surprise, les candidatures à nos offres en cette période se sont raréfiées, les cooptations se sont arrêtées, et les personnes en poste que nous sollicitions se montraient prudentes et peu réceptives à engager un processus de recrutement.

Julien, consultant Zenika à Paris : " J'étais quelque peu inquiet de ce changement d'entreprise, compte tenu des retours de la partie "commerciale". Beaucoup de mes collègues en mission se sont retrouvés du jour au lendemain en chômage partiel, et le ratio offre de mission / candidat avait explosé. L'inquiétude de démarrer dans une nouvelle entreprise en inter-contrat, et avec la possibilité pour l'entreprise de mettre fin facilement à la période d'essai, n'était pas des plus rassurantes. Néanmoins, l'équipe Zenika avait bien avant mon intégration, présélectionné trois missions qui pourraient coïncider avec mon profil et à mes souhaits. Cela m'a rassuré et convaincu de sauter le pas."

Paradoxalement, il était devenu plus simple de contacter de potentiels candidats, la majorité en télétravail, ou même en activité partielle. Cette situation en a amené un grand nombre à répondre favorablement à nos sollicitations. Le fait de ne pas imposer le déplacement sur site a probablement facilité nos prises de contact. Néanmoins, une majorité ne se disait pas prête à envisager le changement.

Le processus de recrutement en full remote a fait son apparition pour un grand nombre d'entreprises je pense. Pour une minorité d'entre elles, avec une équipe internationale ou répartie, il n'y a pas eu de grande transformation. Mais pour la majorité des entreprises, surtout en France avec des approches très latines d'entretiens, il a fallu s'adapter.

Chez Zenika, certains entretiens se faisaient déjà en vision, car certains chargés de recrutement sont responsables de plusieurs agences, souvent éloignées géographiquement. De plus, nous faisions preuve de souplesse, en proposant des entretiens ailleurs que dans nos locaux pour éviter à un candidat le trajet pour venir à un entretien, ou en proposant un échange en visio pour la toute première partie du process. J'ai apprécié la collaboration des hiring managers. Je n'ai pas reçu de contestations particulières ou remarque du type "si je ne peux pas serrer la main du candidat je ne peux pas me prononcer". La transposition des entretiens techniques par nos consultants s'est également déroulée sans problème, bien qu'il ait fallu adapter ou outiller certains exercices comme le pair-programming.

Pour les candidats, les entretiens en visio présentent des avantages certains. Cela désacralise l'aspect très conventionnel de l'entretien : quel est le dress code ? Comment dois-je me tenir, quelle gestuelle adopter ? Cela évite le stress de ne pas trouver les locaux, d'arriver en retard à cause des transports ou d'embouteillages.

Certains recruteurs regrettent de perdre une partie de la communication non verbale qui est effectivement plus importante dans un échange sur place. Mais l'entretien en visio force à se concentrer sur ce que dit l'autre plus que sur comment il le véhicule, sur ses compétences plus que sur un ressenti. Enfin, ces process en remote démontrent que la qualité d'un échange, le fait d'engager et d'embarquer une personne ne tient pas tant au format de l'entretien qu'aux personnes qui mènent ces entretiens.

"J'ai d'abord été surpris par le contact avec Marine, chargée de recrutement pour nos agences de Lyon et Grenoble, qui a été pris très rapidement. J'ai surtout été frappé par son humanité, son écoute et son honnêteté. J'ai enchaîné ensuite les autres phases du recrutement, et là où il m'a fallu insister auprès des concurrents pour effectuer les rencontres par visio, cela s'est fait assez naturellement avec Zenika. Un point qui peut paraître anodin, mais le simple fait qu'il existe un outil de prévu m'a fait penser que c'était la méthode standard de prise de contact.
Personnellement cela m'arrangeait pour différents points: je n'avais pas à faire quarante minutes de trajet, et j'étais simplement plus à l'aise. Au-delà de ça, je ne fais pas vraiment partie des gens qui ont besoin d'être présents physiquement dans un lieu pour m'imprégner de l'atmosphère, donc la visite des locaux ne m'a pas manqué, sachant en plus que je n'allais pas y passer la majeure partie de mon temps.
La suite semble logique. J'ai rencontré pendant chaque étape, différents Z qui m'ont, eux aussi, frappé par leur bienveillance et leur abordabilité." Jordan, consultant Zenika à Lyon
Alison : "Contrairement à d'autres personnes avec qui j'ai pu échanger, j'ai extrêmement bien vécu ces process de recrutement à distance : pas besoin de se stresser pour le trajet et d'arriver trop à l'avance pour être sûr de ne pas être en retard, pas d'histoire de jugement sur la poignée de main,

etc. J'ai même eu l'impression que ça remettait plutôt les recruteurs et candidats sur un pied d'égalité.
Et puis globalement, j'ai trouvé que ça accélérait pas mal les rythmes entre deux entretiens et les réponses. Ce qui est quand même un gros point positif. Mon expérience de recrutement avec Zenika a été très positive. Comme pour un recrutement classique, j'ai été approchée par téléphone, jusque-là, pas de différence notable (si on omet le fait que j'ai accroché tout de suite).
Et puis un premier entretien en visio avec Jean Dupuis & Laurent Tardif (Directeur d'Agence et Directeur Technique de l'agence de Grenoble), où j'ai été reçue comme une humaine et pas un CV à placer. Autant dire que ça m'a aidé à me projeter à bosser avec eux - et pas "pour eux". Honnêtement, je n'ai ressenti aucune différence avec un process en présentiel, si ce n'est que tout le monde était tranquillement chez soi. Finalement, un dernier entretien (toujours en visio) dans la même semaine avec deux autres collègues de l'agence pour voir si le feeling passait et un appel le soir même pour confirmer leur volonté de m'embaucher. En deux semaines, c'était bouclé, quel pied !"

Comment gérez-vous l'onbarding à distance ?
Au-delà du process de recrutement, l'arrivée et l'intégration des nouveaux embauchés en plein confinement ont dû être adaptées également pour se faire à distance. De la livraison du matériel à domicile aux rituels mis en place pour créer du lien avec les nouveaux arrivants, tout a été repensé. Cela n'égalera évidemment pas une expérience où les gens se rencontrent en vrai, dans les locaux et partagent un moment de convivialité moins numérique.
Jordan : "J'ai pu commencer très rapidement, chez un client de longue date, entouré par deux consultants Zenika, qui se sont occupés de mon onboarding, sur un mode hybride sur site et à distance :
Même si j'imagine que les choses devaient avoir une autre ampleur avant, j'ai tout autant l'impression d'avoir rejoint le collectif Zenika. Les différents évènements organisés m'ont aussi permis de croiser de nouvelles têtes. J'ai par exemple enchaîné mon arrivée avec la Technozaure, évènement interne de conférences techniques et de partage"
"Au travers de ces divers processus de recrutement/ onboarding et des projets réalisés à distance et /ou en présentiel, je me suis forgé l'opinion que le télétravail n'impacte pas la cohésion et la performance de l'équipe si tant est que les divers outils de communication à notre disposition soient utilisés de manière judicieuse. J'espère donc qu'une organisation du travail hybride saura émerger lors de la sortie de crise et que le télétravail, ainsi que les processus d'entretien et d'onboarding réalisés à distance, seront acceptés au même titre qu'un mode de fonctionnement plus conventionnel." Alexis.

Et demain ? on continue ou on revient comme avant ?
En ce qui nous concerne, nous conserverons la souplesse que procurent les entretiens en remote aux candidats et aux recruteurs. Ces mois ont démontré que cela n'était en aucun cas un obstacle pour recruter, et à ce jour, tous les entretiens se font encore à 95 % en remote. Nous nous adaptons et si certains candidats nous demandent à venir faire un entretien en présentiel pour voir nos locaux par exemple, nous nous arrangeons. Mais nos candidats continueront à avoir le choix du type d'entretiens qu'ils souhaitent ! De plus, nous voyons que l'engagement des personnes qui nous rejoignent après un process en full remote n'est en rien inférieur à celui qu'avaient les candidats qui prenaient souvent trois fois le temps de se déplacer à nos locaux pour nous rencontrer auparavant. Nous réfléchissons à de nouvelles expériences que nous pourrions proposer à nos candidats, pour garder l'aspect de convivialité et de partage propres à nos valeurs. L'utilisation d'outils pour rendre l'expérience encore plus collaborative, plus interactive et toujours différenciante est en phase de tests. Stay tuned, nous pourrions vous surprendre !
"Pour le futur du recrutement, je suis sûr que le marché ou au moins Zenika, aurait tout à gagner à conserver un mode de recrutement flexible, qui laisserait la possibilité aux candidats qui le souhaitent de l'effectuer à distance. C'est par exemple l'occasion d'attirer des talents excentrés ou de mettre sur un pied d'égalité les interlocuteurs, et il y a certainement tout un pan créatif à explorer. En ce qui me concerne, à aucun moment le distanciel s'est posé en obstacle, et je suis plus qu'enthousiaste de voir comment les choses vont évoluer, que ce soit pour Zenika ou ses clients sur les sujets du travail à distance." Jordan.

Conclusion

Pour notre part, en 2020, l'activité de recrutement a repris dès le mois de juin, pour préparer les embauches de fin d'année, toujours de manière prudente, et dès septembre, l'équipe de recrutement a repris une activité à 100 % pour préparer les embauches de début 2021. Par rapport au plan de recrutement initial de 2020, 50 % des embauches prévues ont été réalisées.
Pour le moment, les prévisions pour 2021 restent prudentes, le Syntec envisage notamment une croissance globale de 1 %. Mais le vent de la reprise se fait sentir. Les entreprises se préparent à recruter, comme en témoignent certains de nos indicateurs. On peut citer le nombre de nouveaux postes de recruteurs et recruteuses tech, essentiels pour trouver les bons profils. Nous prévoyons d'ailleurs l'ouverture prochaine de trois postes pour répondre à notre ambition de développement. Le plan de recrutement prévu pour 2021 retrouve un volume similaire à celui qui était prévu en 2020. De leur côté, les candidats semblent encore attendre une période plus propice au changement. Beaucoup reportent leur réflexion au moins jusqu'à la rentrée prochaine. La méfiance semble encore plus marquée vis-à-vis des ESN qui, faute de missions, n'ont pas hésité à mettre fin aux périodes d'essai en 2020.
En conclusion, c'est principalement le premier confinement qui a eu l'effet le plus visible sur les recrutements. Avec les nombreuses inconnues de la situation, les embauches ont complètement été dépriorisées, au profit des remaniements au sein de l'entreprise. Aujourd'hui, après 1 an de crise sanitaire, ces inconnues font partie du quotidien, et les recrutements peuvent reprendre. Ce qui me semble positif pour le futur, c'est qu'une fois la crise derrière nous, nous continuerons de bénéficier des aménagements mis en place.

TDD sans pitié
(avec l'aimable participation de Kraftos)

Arnaud Thiefaine
Software Crafter
Arolla
Twitter: @ArnaudThiefaine
https://github.com/athiefaine

arolla

Illustrateur : Tanguy Raoul

illustration : Camille Kuo

Préambule

C'est votre premier jour chez Crafteo, une nouvelle startup très orientée nouvelles technologies et bonnes pratiques de l'artisanat logiciel. On vient à peine de vous faire les présentations de vos nouveaux collègues qu'un grand fracas et des cris de détresse retentissent. Quelques instants plus tard, vous voyez passer un des développeurs transportés sur une civière et manifestement mal en point.

Devant votre mine ébahie, Mehdi, qui vous a accueilli, vous explique :

- C'est Franck. Hier il a oublié de reformater le code avant de commiter, et ça n'a pas échappé à Kraftos qui était furax…
- Kraftos ? demandez-vous.
- De son vrai nom Helmut Rodriguez, c'est le team lead. Mais dans l'équipe on le surnomme Kraftos, en rapport avec son intransigeance avec tout ce qui concerne le Craft. D'ailleurs, à l'époque où il était encore un des seuls à packager les applications J2EE à la main, on le connaissait aussi en tant que God of WAR. Comme J2EE est tombé en désuétude, il préfère maintenant se faire appeler Code of War(1).
- Mais c'est quand même très violent ! protestez-vous
- Cela dit, il ne refera pas la même erreur. Je lui avais pourtant répété moult fois de mettre une Save Action dédiée dans son IDE…

La tirade de Mehdi est alors interrompue par un colosse sortant de son bureau. Remarquant la veine qui pulse à son front, vous en déduisez qu'il s'agit de Kraftos. Celui-ci vous remarque, vous toise quelques secondes avant de vous interpeller :

- Ah ! Tu dois être le nouveau ! Tu tombes bien, j'ai besoin de me détendre vu le travail de sagouin que je viens de voir. On va se faire une session de pair-programming, je vais pouvoir voir de quel bois tu es fait.

(1) Les plus geeks d'entre vous auront bien évidemment saisi la référence vidéoludique …

Une sueur glacée vous coule le long de l'échine. Tout à coup, vous n'êtes plus tout à fait confiant dans vos compétences, et vous redoutez de vous faire laminer.
Sentant qu'il serait fort peu approprié de vous soustraire à cette session de binômage, vous décidez de donner le meilleur de vous-même, afin de prouver que vous n'êtes pas un imposteur.
Vous êtes maintenant le héros. A vous de jouer.
Rendez-vous au 100

100

Helmut Rodriguez prend place à côté de vous, et vous explique brièvement la fonctionnalité à développer, qui ne semble pas très compliquée. L'implémentation vous paraît même évidente. Kraftos vous donne le clavier.
Vous pouvez d'ores et déjà proposer votre solution dans le code de production pour impressionner Kraftos (rendez-vous au 101).
Vous pouvez aussi flairer une entourloupe dans cette apparente facilité et envisager une approche plus précautionneuse en vous rendant au 102.

101

Et voilà, en deux minutes, emballé c'est pesé, prêt à partir en production. Vous n'avez pas fini d'écrire la commande "git commit" que Kraftos vous empoigne par le colbac tout en demandant où sont les tests. Tout en vous demandant, de façon purement rhétorique, si ce code était supposé partir en prod sans être testé, il vous plaque violemment au sol et vous finit à coups de genoux dans les chicots. Il vous hurle dessus avec son accent allemand : "Une fois pour toutes, on n'écrit pas de code de production sans avoir de test qui échoue, quand est-ce que ça va finir par rentrer ?! T'es qu'un bon à rien, t'es viré !".
Il ne vous reste plus qu'à ramasser vos dents avant de vous rendre au paragraphe 404. Mais avant cela, profitez de l'occasion pour reprendre les bases du TDD au paragraphe 502.

102

Vous vous dites que ça ne peut pas être aussi simple, et vous esquivez adroitement le piège tendu par Kraftos en ne fonçant pas tête baissée sur la solution. Même si ce qu'on vous demande est simple, il vous paraît important de le tester.
Idéalement, il faudrait même écrire un test bidon sans rapport avec la fonctionnalité à développer, histoire de vérifier que votre environnement de test est correctement configuré. Seulement, à côté de vous, Kraftos semble s'impatienter, et vous craignez qu'il ne s'énerve rapidement.
Peut-être vaut-il mieux directement écrire le code qui teste la fonctionnalité à implémenter, afin d'avancer rapidement et de calmer Kraftos (dans ce cas, rendez-vous au 103).
L'autre option est de vérifier l'environnement de test en prenant un peu plus de temps, au risque de subir les foudres de Kraftos (rendez-vous alors au 104).

103

Au moment où vous achevez d'écrire une assertion pour la fonctionnalité en cours, un violent choc derrière la nuque vous fait lâcher le clavier. Kraftos, visiblement mécontent, vient de vous asséner un taquet dans le cou. Il vous explique

pourquoi : rien ne garantit au préalable que votre framework de test fonctionne correctement et est capable de signaler des tests en échec, il aurait donc fallu jouer un test factice pour le vérifier.
Helmut Rodriguez vous explique ensuite que ce n'est pas gravissime, et que la claque dans la nuque est simplement un avertissement, proportionné à la faible gravité de votre erreur, et offert à titre amical. Vous prenez conscience que vous avez quelque peu irrité Kraftos. Notez ce fait dans votre Backlog.
En sachant que désormais vous allez devoir marcher sur des œufs, rendez-vous au 105.

104

Vous écrivez un test factice, échouant en vérifiant que *false* est différent de *true*. Comme prévu, le test échoue, et Kraftos semble se détendre un peu. Confiant dans votre environnement de test, vous allez pouvoir rentrer dans le vif du sujet en vous rendant au 105.

105

Le test ne compile pas. Vous générez les classes et méthodes du code de production à partir de l'IDE. La méthode générée renvoie *null*. Le code compile maintenant.
Vous pouvez remplacer *null* par la valeur attendue (rendez-vous au 106).
Vous pouvez aussi exécuter le test, même si vous savez pertinemment qu'il va échouer (rendez-vous dans ce cas au 107).

106

Kraftos manque de s'étrangler en vous voyant faire fonctionner le code sans vous être assuré qu'il échouait avant. Vérifiez votre Backlog. Si vous avez déjà énervé Helmut Rodriguez avant, rendez-vous au 108. Sinon rendez-vous au 109.

107

Le test échoue, comme prévu. L'étape suivante est de le faire fonctionner. Pour cela, les possibilités suivantes s'offrent à vous :

- proposer une implémentation juste suffisante pour faire passer le test, au risque que Kraftos juge cela trop naïf ou paresseux : dans ce cas, rendez-vous au 111,
- mettre en place une solution plus élaborée couvrant plus de cas, en d'autres termes montrer que vous savez programmer efficacement : rendez-vous alors au 110.

108

Cette fois-ci, Kraftos a décidé de ne pas laisser passer votre bourde. Il vous attrape, vous soulève et vous projette dans un groupe de bureaux (marquant un Strike au passage en faisant tomber tous les écrans). Bilan pour vous : de multiples écorchures, mais surtout deux côtes cassées. Le comble de tout ça, c'est que Helmut Rodriguez facture les dégâts à votre cabinet de conseil.
Il va vous falloir du temps pour panser vos blessures au fatidique paragraphe 404, temps que vous pourrez mettre à profit pour réviser les bases du TDD (expliquées au paragraphe 502).

109

Helmut Rodriguez vous demande de vous lever de votre fauteuil. Quelque peu hésitant, vous obtempérez, puis il vous gratifie d'un coup de genou en plein dans le creux de la cuisse. C'est très douloureux sur le moment, mais vous allez survivre. Kraftos vous indique qu'il laisse passer pour cette fois, mais qu'il ne sera pas aussi clément à votre prochaine erreur. Notez sur votre Feuille d'Aventure que Helmut Rodriguez est un poil agacé. Conscient que vous progressez désormais sur un terrain miné, vous vous rendez au 107.

110

En une ligne de code, vous arrivez à couvrir la plupart des cas possibles. Un coup dans le dos d'une violence extrême, à décoller le péritoine, vous prouve que ce n'était pas la bonne direction.
Le souffle coupé, vous réalisez que le fait de proposer directement une implémentation trop générique se fait au détriment des cas de tests, et laisse une porte ouverte à l'apparition de régressions. Il n'est pas nécessairement utile de s'appesantir sur les détails du sale quart d'heure que Kraftos vous fait passer. Vous pouvez directement vous rendre au tragique paragraphe 404.

111

Vous écrivez la ligne de code juste suffisante pour faire passer le test au vert. Kraftos vous toise de façon appuyée, comme s'il attendait la suite. Comme vous n'avez pas encore traversé l'open space en vol plané, c'est presque de bon augure. Mais c'est aussi le signe que vous ne devez pas encore vous relâcher, et décider très soigneusement de votre prochaine action :
- vous pouvez généraliser votre implémentation pour couvrir plus de cas en vous rendant au 112,
- mais peut-être serait-il préférable de prendre votre temps, en écrivant au préalable un second exemple de test pour la même règle de gestion ; dans ce cas, rendez-vous au 113.

112

C'est en recrachant la moitié des touches du clavier (ainsi que quelques-unes de vos dents) que vous réalisez que Helmut Rodriguez ne supporte pas qu'on grille les étapes. Et quelque part, vous ne pouvez pas lui en vouloir vu l'entrain avec lequel vous avez manifestement choisi d'ignorer la première règle du TDD : on ne modifie pas le comportement existant avec des tests au vert sans écrire un cas de test qui échoue au préalable. La situation n'étant pas rattrapable, il ne vous reste qu'à vous rendre au délétère paragraphe 404.

113

Constatant qu'après avoir écrit un test qui échoue en vérifiant un second exemple, il vous reste toutes vos dents, vous réalisez que ce choix s'avère payant. Cette approche est pertinente également parce qu'elle permet de s'assurer que l'implémentation proposée au préalable ne fonctionne pas grâce à un coup de chance, et permet de s'assurer que le code est testé de façon exhaustive (et de se prémunir de certaines

régressions). Le répit est de courte durée, parce que Helmut Rodriguez vous met la pression pour la suite. Vous sentez que vous n'avez pas intérêt à le décevoir.
Vous pouvez proposer l'implémentation la plus simple et naïve possible couvrant spécifiquement vos deux cas de tests. Le risque est que Kraftos pense que vous vous moquez de lui et qu'il vous le fasse payer très cher. Si c'est néanmoins votre choix, rendez-vous au 114.
Vous pouvez aussi aller directement à une implémentation évidente qui serait plus générique : rendez-vous alors au 115.

114

Vous poussez un soupir de soulagement : Kraftos semble satisfait de votre approche. Les tests sont au vert. Mais vous n'êtes pas encore tiré d'affaire. Quel sera votre prochain choix :
- Passer à une nouvelle règle métier ? Vous saurez si cette décision est judicieuse en vous rendant au 116.
- Prendre le temps de refactorer le code de production ? Vous connaîtrez les conséquences de cette approche en vous rendant au 117.

115

Vous tentez de fournir directement la solution couvrant les 2 cas de tests. Manque de chance, les 2 tests échouent, et vous voyez Helmut Rodriguez se crisper. La situation semble très mal engagée, et vous appréhendez l'imminent retour de flammes.
Vous pouvez tenter de sauver la situation en corrigeant le code. Vous savez que vous vous n'avez pas intérêt à échouer, mais si tel est votre choix rendez-vous au 119.
Vous pouvez aussi chercher à vous épargner des souffrances en essayant de prendre la fuite ; dans ce cas, courez au 120.

116

Kraftos laisse échapper un sifflement, qui vous alerte sur le fait que vous allez peut-être un peu trop vite en besogne. Les tests étant au vert, c'était l'occasion idéale pour améliorer la qualité du code avec un bon filet de sécurité.
Il semble que vous vous soyez mis dans une situation délicate. Si c'est votre première bourde, il y a une chance pour que Kraftos laisse passer (rendez-vous au 118 pour avoir une idée de ce qui vous attend).
Mais il se peut que vous soyez déjà en train de commettre votre seconde ânerie : il va sans dire que vous allez vous faire ouvrir très copieusement. Épargnez-vous les détails sanglants en allant directement à l'imbuvable paragraphe 404.

117

Vous optez pour une approche par triangulation : le code répondant spécifiquement à plusieurs exemples de la même règle de gestion, vous généralisez pour réduire la duplication. Kraftos approuve pleinement. Avant de partir déjeuner, il vous explique que dans l'après-midi c'est lui qui prendra le clavier.
Rendez-vous au 200.

118

Helmut Rodriguez vous explique que c'était le moment de généraliser le code, et en profite pour vous montrer la technique de triangulation (qui consiste à remplacer les exemples en dur dans le code qui répondent aux tests par une implémentation plus générique). Bien sûr, la pédagogie de Kraftos n'est pas gratuite, et il vous offre une pénalité, sous la forme d'un point de pression très douloureux sur la clavicule, dont vous vous souviendrez longtemps. À votre grand soulagement, Kraftos est ensuite appelé sur un problème de prod.
Rendez-vous au 200.

119

Vous décidez d'affronter votre destin. Vous allez vous faire salement éclater, c'est sûr, mais au moins Kraftos va vous expliquer par le détail et insultes en Allemand où se situent vos errances.
Pour faire simple, vous avez sauté une étape importante du TDD, en cherchant à refactorer alors que les tests étaient au rouge. En procédant ainsi, vous vous empêchez de revenir facilement à un état stable, ce qui va aggraver la situation et vous faire perdre un temps précieux. Il aurait été préférable de revenir à la dernière situation stable connue plutôt que de vous entêter.
Il ne vous reste plus qu'à vous rendre au préjudiciable paragraphe 404.

120

Avant même que Kraftos n'ait pu vous agripper, vous vous aplatissez au sol, enchaînez sur une roulade et vous mettez à courir comme un dératé. Arrivé à l'escalier, vous n'arrivez pas à ralentir suffisamment et trébuchez. Vous parvenez en bas couvert de contusions, mais vous vous en tirez bien par rapport au sort que Kraftos vous réservait. Ce dernier n'a cependant pas fait l'effort de vous pourchasser.
Vous ne savez pas trop pourquoi vous avez échoué, mais pour le savoir il aurait fallu affronter le courroux de Kraftos : s'il est généreux dans la distribution de fractures et de bosses, il l'est également dans le savoir qu'il dispense conjointement.
Il ne vous reste plus qu'à vous rendre au désolant paragraphe 404.

200

Kraftos s'en est allé vaquer à d'autres occupations, ce qui vous offre un court répit. Vous avez survécu à cette session de pair-programming sur le fil de rasoir sans vous faire massacrer, ce qui semble indiquer que vos connaissances en TDD sont plutôt solides (à moins que vous n'ayez été très chanceux ou très persévérant). Félicitations, vous vous en êtes sorti !

Pour la suite de vos aventures, vous pouvez continuer avec Helmut Rodriguez dans des conditions qui mettront à l'épreuve votre intégrité physique, ou bien chercher à rejoindre des sociétés qui permettent de pratiquer le Craft de façon sereine et bienveillante (par exemple Arolla).

Mais ceci est une autre histoire...

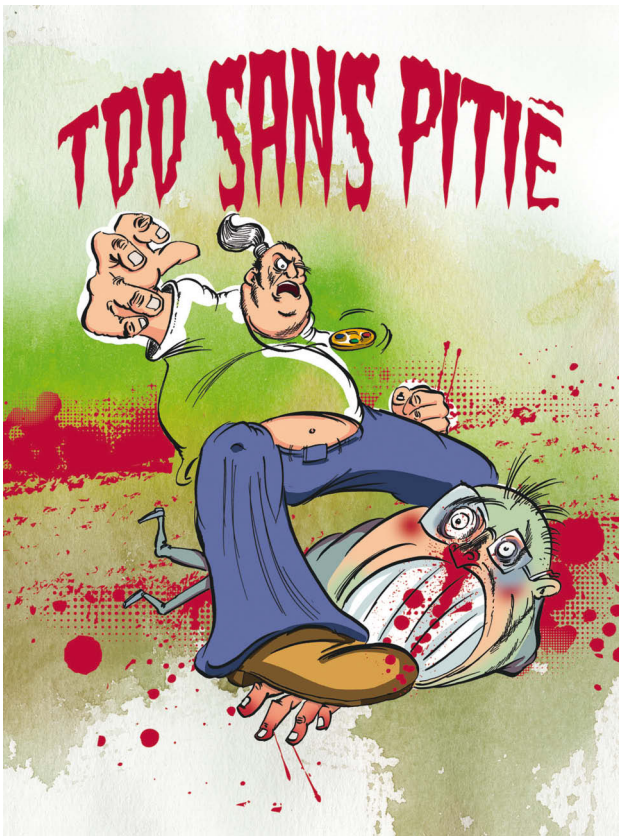
404

Si vous êtes ici, c'est que votre pratique du TDD a fortement déplu à Helmut Rodriguez, et il vous l'a fait savoir à sa façon, très expressive.

Si ça peut vous consoler, sachez que nous ne cautionnons pas l'attitude rugueuse de Kraftos. Le personnage ici dépeint, très caricatural pour les besoins en intensité dramatique du récit n'a bien évidemment pas sa place dans un environnement Craft et n'est bien heureusement pas représentatif de la communauté Craft.

Kraftos, en tant que Crafter confirmé, aurait dû vous expliquer vos erreurs avec bienveillance et chercher à vous faire progresser, plutôt que de réagir en vous traumatisant.

Maintenant, même si Kraftos aurait pu vous l'expliquer de façon plus douce, il semble que votre pratique du TDD n'est pas encore totalement maîtrisée. N'hésitez pas à en réviser les principes au paragraphe 502, avant de tenter à nouveau votre chance.



502

Les bases du Test-Driven Development

Dans l'approche de développement pilotée par les tests (TDD pour Test-Driven Development), on écrit les tests avant d'écrire le code de production. On procède itérativement et par petites étapes (ce qu'on appelle les *baby steps*). Pour gagner en efficacité, l'approche se doit d'être très rigoureuse. Pour cela, on doit suivre un certain nombre de règles, édictées par Robert C. Martin et connues comme les 3 lois du TDD :

- On ne peut pas écrire de code de production tant qu'on n'a pas écrit un test qui échoue.
- On ne peut pas écrire plus d'un test que nécessaire pour échouer, et un test qui ne compile pas est considéré comme un échec du test
- On ne peut pas écrire plus de code de production que nécessaire pour que le test actuellement en échec réussisse

Lorsque les tests sont en succès (au vert), on peut partir sur l'ajout d'un nouveau test. On peut aussi chercher à améliorer le code de production (ou le code de test, si besoin) en faisant un refactoring.

Autrement dit, les seuls moments où on s'autorise du refactoring se situe lorsque les tests sont au vert.

Lorsque les tests sont au rouge, on doit chercher à revenir le plus rapidement possible dans un état stable, et pour cela on fournira l'implémentation la plus simple permettant de passer les tests.

Si on doit modifier le comportement, il est crucial d'ajouter un nouveau test permettant de vérifier ce nouveau comportement.

Le processus TDD se répète donc en plusieurs cycles comportant les étapes suivantes :

1. Écriture d'un test qui vérifie une partie du problème à résoudre ;
2. Ce test doit échouer, montrant que le comportement faisant fonctionner le test n'existe pas dans le code de production ;
3. Écriture du code de production suffisant pour que le test réussisse ;
4. Le test doit passer, ainsi que les autres tests existants ; si ce n'est pas le cas, on repasse sur l'étape 3 ;
5. Refactoring du code si nécessaire : on le rend plus propre sans modifier le comportement.

Les bénéfices de l'approche TDD sont nombreux :

- On se focalise en premier lieu sur le comportement et sur le besoin plutôt que sur l'implémentation.
- Le code de production est totalement testé, il est donc plus fiable et de meilleure qualité.
- Le code de production est juste suffisant pour répondre au besoin, il reste donc relativement simple, sans portions inutiles, et facile à lire (ce qu'on appelle aussi le design émergent).

Les tests servent donc à guider la conception, mais aussi à fournir au fil du temps un filet de sécurité exhaustif contre les régressions.

DocFX : générer votre documentation

Le paradoxe du développeur : d'un côté, il attend des frameworks et outils bien documentés et de l'autre, il rechigne à écrire cette documentation. La documentation est un élément essentiel à tout projet. Et quand je parle de documentation, je parle de tout ce qui touche à la documentation : documentation du code en tant que tel, documentation pour le développeur, documentation pour l'utilisateur final. On oublie souvent qu'il existe des documentations techniques et non une seule doc.

Laurent Ellerbach
Principal Engineer Manager
Microsoft
laurelle@microsoft.com
https://github.com/Ellerbach

L'idée est également de pouvoir la gérer le plus facilement possible sans spécialement avoir besoin d'outils qui soient à l'extérieur du repository de code. Je vais décrire dans cet article des approches pour générer de la documentation automatiquement depuis le code source, et aussi la création de cette doc depuis le repository de code. Mon outil de prédilection est DocFX (cf. https://github.com/dotnet/docfx). C'est un des outils pour générer toute la documentation de https://docs.microsoft.com.

Les outils compagnons mentionnés, des documents complémentaires, les exemples de pipeline se trouvent sur mon GitHub : https://github.com/Ellerbach/docfx-companion-tools

Toute ma documentation est dans mon repository

La documentation, c'est du code. Comme tout code, il doit se trouver dans le même repository que le reste et être soumis aux mêmes règles : PR reviews, linter (vérificateur de syntaxe et de style), pipeline de quality assurance. Si vous n'êtes pas habitués à ces acronymes et termes : le même niveau d'exigence de revue de code et d'assurance qualité doivent être appliqués à la documentation.

Dans tous les projets que je fais avec des clients en co-engineering, c'est un élément sur lequel j'insiste toujours même quand le client ne veut pas de la documentation dans le périmètre du projet. L'autre élément sur lequel j'insiste est la génération automatique de toute cette documentation. L'intérêt est de pouvoir extraire les commentaires (les /// en C# par exemple), et de pouvoir transformer des fichiers Markdown en HTML, mais aussi gérer automatiquement tout ce qui est autour : la génération de la navigation (Table of Content – TOC), générer des structures multilingues, la génération de la documentation depuis le code source, vérifier l'intégrité des documents et liens. Une structure de repository avec la documentation peut ressembler à cela : **Figure 0**

Nous partirons sur cet exemple pour la suite. Cet exemple est simplifié pour aller plus vite. Ce que l'on voit, ce sont deux entrées de documentations : une *devdocs* et une *userdocs*. Les sources qui se trouvent dans *src* seront également générées en tant que documentation.

Intégration avec Azure DevOps et GitHub

Pour ceux qui utilisent Azure DevOps comme repository, il est possible de créer les wikis comme des wiki as code et de les traiter avec des pipelines pour vérifier la qualité des fichiers Markdown générés. Nous verrons cela un peu plus loin. Lorsque vous créez votre projet, vous pouvez ajouter un wiki : **Figure 1.** Choisissez de créer un code as wiki : **Figure 2**

Le wiki apparaîtra comme un folder dans votre projet. À partir de là, vous pouvez appliquer toutes les bonnes pratiques habituelles : linter pour la qualité des fichiers Markdown produits, par exemple, lors d'un PR. Je recommande de désactiver la modification du wiki sans passer par des PR. Cela limite grandement l'intérêt, car les pipelines ne pourront empêcher un merge en cas de malformation des fichiers Markdown par exemple. Et cela limite également l'intérêt des revues de PR. De la même façon, il est possible d'avoir une intégration similaire avec GitHub une fois un wiki créé. Par exemple :

```
git clone git://github.com/vous/votre_proj
cd votre_proj
git remote add -f wiki git://github.com/vous/votre_proj.wiki
git merge -s ours --no-commit --allow-unrelated wiki/master
git read-tree --prefix=wiki/ -u wiki/master
git commit -m "Github wiki subtree merged in wiki/"
```

Évidemment, vous pouvez ajuster les noms des répertoires. Ensuite, comme pour Azure DevOps, je recommande de systématiquement passer par des PR pour que les pipelines d'assurance qualité se fassent.

Vous pouvez sur Azure DevOps créer autant de wiki as code que vous voulez. Azure DevOps générera également des fichiers *.order* que nous allons réutiliser plu tard. Il crée également un répertoire *.attachements* dans lequel tous les fichiers joints, images et autres éléments sont stockés. Nous réutiliserons également cet artéfact dans notre code.

DocFX : the great, the good and the ugly

Aucun outil n'est parfait et DocFX étant un outil, il n'est pas parfait non plus. Il a de très nombreuses fonctionnalités qui le rendent parfait comme la transformation de fichiers Markdown en fichier HTML avec la possibilité de personnaliser et de créer ses propres template.

DocFX permet également de construire les projets .NET et d'extraire les commentaires /// des fichiers pour construire une documentation statique du code. C'est ce que vous avez dans les documentations officielles de Microsoft.

Voici un exemple de génération de ce type de documentation .NET nanoFramework https://docs.nanoframework.net/api utilisant ce mécanisme : **Figure 3**

L'autre intérêt est de pouvoir donner directement à DocFX un swagger et il générera de la même façon la documentation associée. Un swagger est un fichier json qui décrit l'API. La plupart des API publiques en génèrent un automatiquement basé sur le code. Tous les langages et plateformes modernes

Figure 0

```
/userdocs
 /.attachments
  picture-en.jpg
  picture-de.jpg
  photo.png
  otherdoc.pptx
 /en
  index.md
  /plant-production
   morefiles.md
   and-more.md
 /de
  .override
  index.md
  /plant-production
   morefiles.md
   and-more.md
 index.md
 toc.yml
/devdocs
 /.attachments
  Some_attachement.png
  architecture.drawio
 /a_folder
  .override
  info.md
  more.md
 index.md
 .order
/src
 your_code_here
.gitignore
other_root_files
```

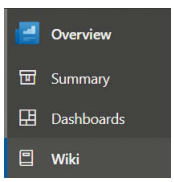


Figure 1

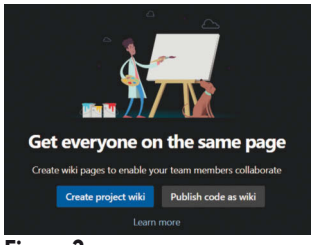


Figure 2

offrent aujourd'hui cette possibilité nativement ou à travers des paquets installables. Le résultat est similaire à celui de la génération de la documentation depuis la source.

À noter qu'il est aussi possible de générer des PDF de documentation également. Il existe une intégration avec Visual Studio 2019. Il est possible d'ajouter ses propres transformations, d'étendre DocFX et d'utiliser des outils tiers pour générer de la documentation à partir de sources type JavaScript.

Le paramétrage s'effectue à l'aide d'un fichier json. Nous verrons un exemple plus loin.

L'autre intérêt est que DocFX se met facilement dans un pipeline (Continuous Integration et/ou Continuous Delivery). Un exemple concret se trouve dans celui utilisé pour construire les docs de .NET nanoFramework : https://github.com/nanoframework/nanoframework.github.io, fichier azure-pipelines.yml.

Ce qui est intéressant dans cet exemple, c'est de voir le nombre de repositories qui sont clonés, construits. Et surtout une fois ce travail effectué, les modifications apportées au repository lui-même sont reprises.

DocFX peut également générer une partie de la table des contenus (TOC). Et c'est là, pour nos scénarios, l'outil devient un peu limité. Dans le cas de mon dernier projet, nous avions des milliers de fichiers Markdown à gérer et à intégrer avec de nouveaux fichiers à créer, changer la structure.

Markdownlint mon amour

La première difficulté a été de nettoyer ces fichiers. En effet, Markdown, comme C#, Java ou tout autre langage, suit des règles. Elles ne sont pas très compliquées. Une bonne liste pour commencer :

https://github.com/DavidAnson/markdownlint/blob/main/doc/Rules.md

Ces règles sont des bonnes pratiques : 1 seul titre principal #, une structure propre de titres ##, ###, chaque titre doit être unique, chaque paragraphe est séparé par une ligne vide, les énumérations doivent être dans le même paragraphe, pas de HTML dans les fichiers, chaque bloc de code doit avoir un nom en minuscule et quelques autres de ce type. Force est de constater que ces règles sont mal connues de la plupart des développeurs ! La mise en place d'un linter, dans ce cas Markdownlint permet de bloquer la fusion de branche lors de PR si un fichier Markdown est mal formé.

Avoir des fichiers bien formés est le gage d'un site généré propre avec les TOC qui sont générées pour chaque document de façon automatique et dynamique. Cela permet aussi de s'assurer que les fichiers générés appliqueront le style (au sens css du terme) lors de la transformation. Partir d'un existant de quelques milliers de documents à migrer et à nettoyer, même avec le linter qui aide et corrige les problèmes, cela prend beaucoup de temps ! Dans notre cas : 2 jours pour une personne ! Mais c'est nécessaire pour la qualité globale.

Le linter permet ensuite de garantir que tout ce qui est mergé soit propre. Conseil : à mettre en place sur l'ensemble du repository, pas juste sur les wiki as code. Côté des règles, vous pouvez choisir celles que vous voulez assouplir ou modifier. De mon côté, je suis stricte, la seule que j'autorise, c'est d'avoir plus de 80 caractères sur une ligne. Cette règle n'existe que pour la lisibilité, pas pour des raisons techniques.

Voici le Yaml nécessaire pour faire fonctionner Markdownlint dans un pipeline Azure DevOps. À ajouter en fonction de votre chaîne de CI/CD. C'est simple et qu'il ne faut surtout pas s'en priver !

```yaml
# Scan markdownfiles on style consistency
- job:
  displayName: 'Execute Markdownlint'
  steps:
    - bash: npm install -g markdownlint-cli
      displayName: 'Install markdownlint'

    - bash: markdownlint -c $CONFIGFILE $WORKDIR
      env:
        WORKDIR: $(System.DefaultWorkingDirectory)
        CONFIGFILE: $(System.DefaultWorkingDirectory)/.markdownlint.json
      displayName: 'Run markdownlint'
```

Gérer les fichiers attachés et images

Pour garder un système de documentation propre, il convient d'avoir un répertoire unique dans lequel stocker tous les fichiers attachés, les images et autres documents. C'est une bonne pratique, cela permet de garder une structure de fichier où il n'y a que des fichiers Markdown et ceux utilisés pour générer la documentation.

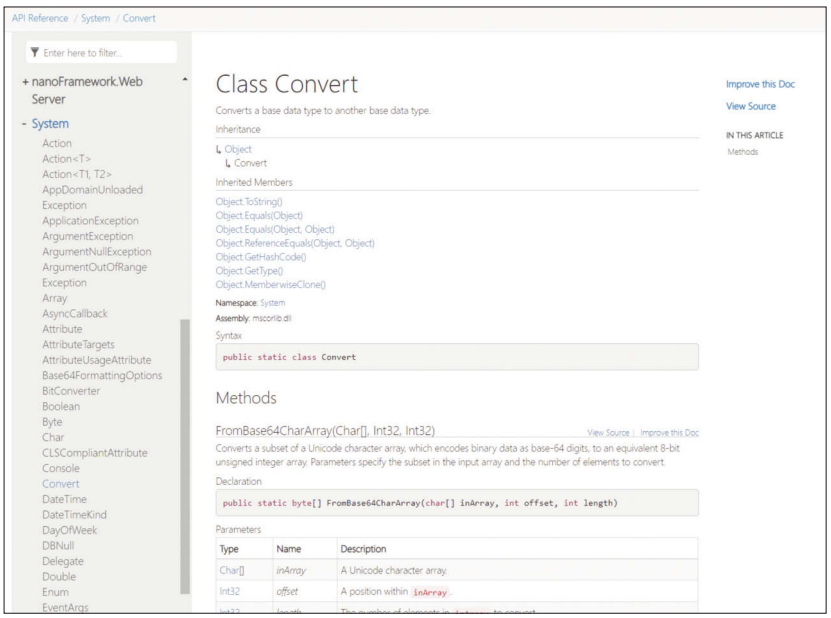
Le processus de revue de PR devrait être suffisant pour garantir que les fichiers non Markdown soient bien dans le répertoire *.attachements*, dans notre cas. Mais cela n'est pas suffisant pour garantir l'intégrité des liens.

Lors de la génération des documents, DocFX donne des warnings sur les liens potentiellement cassés, mais pas sur les documents n'ayant aucun lien. Pour cela, nous avons écrit un outil vérifiant l'intégrité des documents, des liens, les corriger automatiquement et aussi de nettoyer les documents non référencés. Comme pour le linter, il est important de le mettre dans un pipeline. Les sources, le pipeline sont disponibles dans le repository d'exemple.

Génération automatique de TOC

Gérer la table des contenus manuellement est faisable pour un petit nombre de pages. Au-delà de quelques dizaines de pages, l'opération se complique, et j'en ai fait l'expérience.

Figure 3



DocFX sait générer une partie de la TOC, mais ne permet pas de gérer l'ordre des documents par exemple.
De façon à avoir une TOC vivante et dynamique, l'idée est de pouvoir utiliser les titres de chaque document comme entrée, et de pouvoir gérer l'ordre des documents dans la TOC. Pour l'ordre des documents, nous utiliserons le *.order* d'Azure DevOps. Quant à l'entrée dans la TOC, ce sera en ouvrant chaque fichier et en récupérant le titre d'ordre principal. Comme nous sommes sûrs que les fichiers sont propres grâce au linter, cela ne posera aucun problème.
La TOC peut être générée pour l'ensemble d'un sous répertoire. Les TOC générées ne sont pas stockés dans le repository ou alors uniquement comme exemple. À l'instar des autres outils, il est ajouté dans le pipeline de génération de document. L'outil permet aussi de renommer des fichiers. C'est utile pour les noms des répertoires.
Avec l'outil **TocDocFxCreation** on obtient le résultat suivant :

```yaml
# This is an automatically generated file
items:
- name: Digital Manufacturing Platform DMP
  href: README.md
- name: Getting started
  href: getting-started/README.md
  items:
  - name: Getting Started
    href: getting-started/README.md
  - name: Step by Step for ultimate beginner with DMP
    href: getting-started/step-by-step-beginner.md
  - name: Working in the VDI
    href: getting-started/working-in-the-vdi.md
- name: Working agreements
  href: working-agreements/README.md
  items:
  - name: Working Agreements
    href: working-agreements/README.md
  - name: Estimations
    href: working-agreements/estimations.md
  - name: Definition of Done
```

Documentation multilingue

Il existe plusieurs patterns pour générer de la documentation multilingue. Celui que je préfère est un répertoire par langue avec un fichier traduit par langue. Cela permet de faire des redirections simples. Par exemple https://docs.microsoft.com/ redirigera sur la langue préférée de l'utilisateur. Les liens peuvent être donnés sans le paramètre de langue.
Le défi est de toujours avoir les mêmes documents dans toutes les langues et une structure identique. Un changement dans une langue doit être répercuté partout : les liens, la création de nouvelles pages, etc. Pour les liens, les fichiers attachés, nous pouvons réutiliser les outils précédemment mentionnés.
Une TOC par langue est générée. Le nom des répertoires peut être traduit proprement dans chaque langue et déposé dans le fichier *.override*.
Rappel : il est important de garder la structure de la documentation dans chaque langue. Pour cela, nous avons développé un outil dédié : **DocLanguageTranslator**. L'outil va regarder dans chaque répertoire les fichiers Markdown et la structure des répertoires. Cet outil va permettre à la fois de vérifier que la structure soit intègre. Il est à utiliser dans un pipeline. **Figure 4**
L'outil fait appel aux Azure Cognitive Services (https://azure.microsoft.com/services/cognitive-services/). L'outil et tout son code source sont disponibles dans le même repository comme tout le reste.

Conclusion

Maintenant, si nous mettons bout à bout tous les éléments, on arrive à automatiser et à faciliter la création de documentation à la fois pour les développeurs et les utilisateurs finaux. C'était notre objectif de départ.
L'expérience peut ressembler à cela : **Figure 5**
Vous noterez différentes entrées dans le menu principal, un style différent de celui par défaut. L'entrée *.NET Reference* contient toute la documentation extraite du code .NET, *Frontend* de façon similaire, mais avec Angular extraite du code avec typedoc. *Developers* contient la documentation pour les développeurs. Et *User documentation* pour les utilisateurs. Chaque langue possède sa TOC. L'ensemble des outils et exemples sont accessibles sur notre GitHub : https://github.com/Ellerbach/docfx-companion-tools
Au moment où j'écris ces lignes, un PR est en cours de validation pour ajouter ces éléments à la documentation de DocFX. J'espère vous avoir convaincu que créer et gérer sa documentation comme son code est important. Et que ce n'est forcément plus difficile que d'écrire du code.

Figure 5

```
Translating C:\DMP\userdocs\de\Data-szie-estimation.md
Translating
.....................................................................................
Saving C:\DMP\userdocs\fr\Data-szie-estimation.md
Translating C:\DMP\userdocs\de\index.md
Translating
..
Saving C:\DMP\userdocs\fr\index.md
Translating C:\DMP\userdocs\de\plant-production\example.md
Translating
....
Saving C:\DMP\userdocs\fr\plant-production\example.md
Translating C:\DMP\userdocs\de\plant-production\second-file.md
Translating
..
Saving C:\DMP\userdocs\fr\plant-production\second-file.md
Translating C:\DMP\userdocs\de\plant-production\another-dir\another\and-another\something.md
Translating
..
Saving C:\DMP\userdocs\fr\plant-production\another-dir\another\and-another\something.md
Process finished. 5 translated and properly created. Please make sure to run the Markdown linter and also check the file links
and images.
```

Figure 4

Benoît Prieur

Développeur indépendant pour sa société Soarthec. Il est l'auteur de plusieurs livres à propos de développement logiciel publiés aux éditions ENI, dont un ouvrage récent à propos de PyQt5, paru en janvier 2021.

PyQt5 : développement d'une application de téléversement d'images vers Wikimedia Commons

Le propos de cet article est de présenter l'excellent framework PyQt, dans sa version 5. Cet éclairage prendra comme fil conducteur le développement d'une application de bureau de téléversement d'images vers Wikimedia Commons, la médiathèque en ligne des projets Wikimedia.

1. PyQt, binding de Qt pour Python

À l'origine il y a donc Qt. Qt est un framework de développement d'applications graphiques en C++, offrant des composants graphiques relatifs à de nombreux aspects : l'interface bien sûr, mais également l'interaction avec des bases de données, la gestion des couches basses, une gestion fine des threads, etc.

Le succès de ce framework a induit le développement de passerelles logicielles, des bindings, qui permettent d'utiliser Qt avec un autre langage que le C++. Ainsi il existe Qt# pour les langages .Net, Qt Jambi pour Java, QtRuby pour le langage Ruby et donc PyQt pour le langage Python.

Signalons à ce propos un autre binding pour Python, nommé PySide : ce binding est principalement développé par Nokia et est distribué sous une licence plus permissive que celle de PyQt. En effet, PyQt est distribué sous une double licence GNU/propriétaire alors que PySide est en LGPL. PyQt5 est la version 5 du framework. La première version de PyQt6 a été publiée en janvier 2021, que nous n'utilisons pas pour cet article.

Travailler avec PyQt implique de se référer à la documentation Qt, écrite pour le développement C++. Ici se trouve probablement la principale difficulté du développement PyQt : lire et utiliser une documentation prévue pour le développement C++, à des fins de développement Python.

La documentation en ligne de Qt : https://doc.qt.io/

2. Développement d'un logiciel de téléversement d'images vers Wikimedia Commons

Wikimedia Commons est la médiathèque en ligne de l'encyclopédie Wikipédia et plus généralement des projets hébergés par la fondation Wikimedia. Cette médiathèque contient près de 70 millions de fichiers médias (essentiellement des photos, mais aussi divers types d'images, des vidéos et des sons). Ces documents sont utilisés pour illustrer les projets Wikimedia, mais peuvent également être utilisés à l'extérieur de cet écosystème. Tout à chacun peut réutiliser ces documents dans ses propres publications sous réserve du respect de la licence libre indiquée, en général issue de Creative Commons.

Ainsi les contributeurs et contributrices de Wikimedia Commons, des wikiphotographes notamment, prennent des photographies, les post-traitent puis les téléversent sur le projet en précisant un certain nombre de méta-données. Cette dernière étape peut être fastidieuse surtout si un grand volume de photographies doit être envoyé sur Wikimedia Commons. De ce fait, l'interface web de téléversement n'est pas toujours appropriée.

Il y a plusieurs alternatives permettant de gagner du temps, en particulier sur la mise en place des métadonnées associées à une photographie. L'un des plus connus et utilisés est le logiciel de bureau codé en Java, nommé Commonist. Son développement en Java lui permet d'être utilisable sur tous les systèmes d'exploitation, Windows, Linux ou macOS. Notre logiciel codé en PyQt/Python pourra lui aussi être portable sur ces différents systèmes d'exploitation.

Commonist permet de factoriser judicieusement les méta-données communes à plusieurs images ; il permet également de choisir sa licence de publication d'images et surtout de téléverser vers n'importe quel wikiprojet de la fondation Wikimedia.

- Le wikiprojet Wikimedia Commons : https://commons.wikimedia.org/
- L'interface web de téléversement de Wikimedia Commons : https://commons.wikimedia.org/wiki/Special:UploadWizard
- Le logiciel Commonist : https://commons.wikimedia.org/wiki/Commons:Commonist/fr

Notre projet PyQt5, que nous baptiserons PyCommonist dans la mesure où ses fonctionnalités sont très inspirées de celles de son grand frère codé en Java, sera simplifié dans plusieurs de ses objectifs. En particulier : il ne sera possible de téléverser des images que vers Wikimedia Commons (commons.wikimedia.org) et non vers d'autres wikis (fr.wikipedia.org, hy.wiktionary.org, etc.) comme on le peut avec Commonist.

3. Le maquettage de PyCommonist

On cherche donc ici à développer en PyQt/Python un logiciel similaire à Commonist (Java) par ses fonctionnalités, en en simplifiant quelques-unes (import uniquement vers Wikimedia Commons). Le schéma suivant indique les différents champs qui composent l'application PyCommonist.

Figure 1

La fenêtre est divisée en trois zones grâce à l'usage d'un splitter vertical et d'un autre horizontal.

Zone 1

Cette zone contient :

- une zone de texte pour le nom d'utilisateur Wikimedia Commons.
- une zone de texte pour son mot de passe,

- une zone de texte pour indiquer l'auteur de la série de photos (éventuellement avec le formalisme de template de Wikimedia Commons).
- la licence libre de la série de photos (également avec le formalisme de template de Wikimedia Commons).
- la liste des catégories générales de la série de photos (les noms de catégories sont séparés par un symbole "|").
- deux checkboxes de sélection ou désélection totale des photos à téléverser ainsi que le bouton qui lance l'import.

Voici le rendu de la **Zone 1**. **Figure 2**

Zone 2

- l'utilisateur sélectionne un répertoire sur son ordinateur grâce à un arbre du système local dans lequel il peut sélectionner un répertoire donné du disque local.

Voici le rendu de la **Zone 2**. **Figure 3**

Zone 3

Cette zone contient, pour chaque photo du répertoire sélectionné :

- une checkbox pour indiquer si on désire la téléverser ou non.
- une zone de texte pour le nom de la photo.
- une zone de description qui sera concaténée avec la description générale.
- une liste de catégories qui seront ajoutées aux catégories générales (là encore on les sépare avec le symbole "|").
- la date et l'heure de la photo extraites automatiquement depuis les données EXIF de la photo (quand cela est possible).
- la latitude, longitude et direction (DSC, heading) de la photo extraites automatiquement depuis les données EXIF de la photo (quand cela est possible).

Voici le rendu de la **Zone 3**. **Figure 4**

Précisons qu'une statusbar est située tout en bas de la fenêtre sous les zones 2 et 3 : les messages à destination de l'utilisateur sont écrits dans cet emplacement. Voici alors le rendu global de l'application PyCommonist : **Figure 5**

4. Le développement de l'application en Python/PyQt5

4.1 La définition de l'application PyQt5

On commence par créer un fichier main.py qui utilise la classe PyQt QApplication et qui lance la fenêtre, elle-même qui est une instance de notre classe PyCommonist.

```
import sys
from PyQt5.QtWidgets import QApplication
from PyCommonist import PyCommonist

def main():
    app = QApplication(sys.argv)
    ex = PyCommonist()
    sys.exit(app.exec_())

if __name__ == '__main__':
    main()
```

- QApplication dans la documentation PyQt5 : https://doc.qt.io/qt-5/qapplication.html

Cette classe PyCommonist hérite de la classe PyQt nommée QWidget qui permet d'octroyer à la classe fille PyCommonist les propriétés élémentaires d'un widget PyQt.

```
class PyCommonist(QWidget):

    tool = None

    def __init__(self):
        super(PyCommonist, self).__init__()
        self.initUI()
```

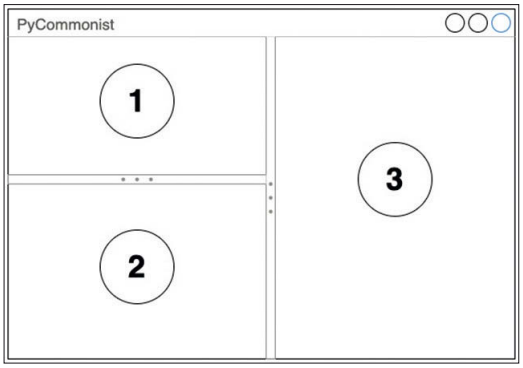


Figure 1

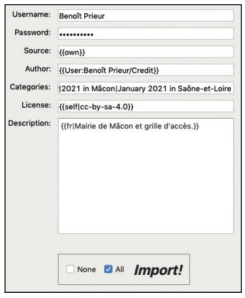


Figure 2

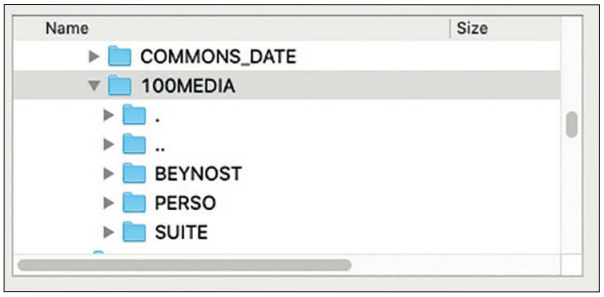


Figure 3

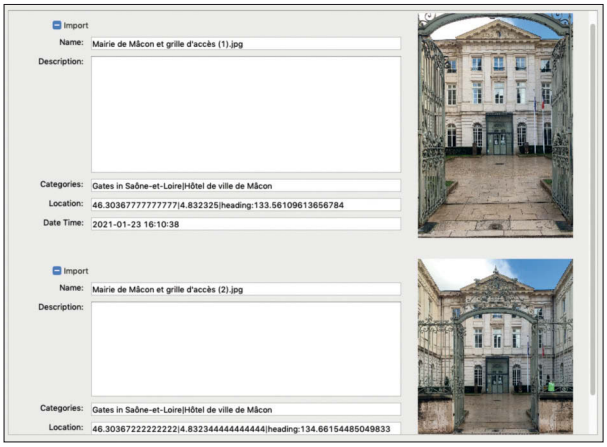


Figure 4

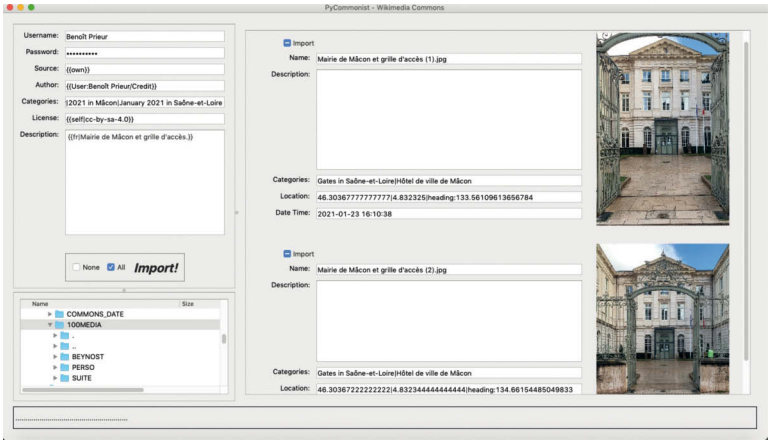


Figure 5

```
    self.threads = []
    self.workers = []


def initUI(self):

    self.currentDirectoryPath = ''

    self.generateSplitter()
    self.generateLeftTopFrame()
    self.generateLeftBottomFrame()

    self.showMaximized()
    self.setWindowTitle('PyCommonist - Wikimedia Commons')
    self.show()
```

- QWidget dans la documentation PyQt5 : https://doc.qt.io/qt-5/qwidget.html

4.2 La définition des splitters vertical et horizontal

On utilise pour cela un contrôle de disposition vertical QVBoxLayout et un contrôle de disposition horizontal QHBoxLayout. Les contrôles de disposition permettent d'agencer, ici verticalement et horizontalement, les différents composants graphiques PyQt que l'on nomme widget. QScrollBar, QSplitter, et QFrame sont des widgets. Les contrôles de disposition nous permettent d'agencer les différentes frames (QFrame) qui correspondent aux zones 1, 2 et 3. On définit les barres de défilement de chaque zone (QScrollBar). Concrètement les splitters sont définis grâce à la classe QSplitter.

- QVBoxLayout dans la documentation PyQt5 : https://doc.qt.io/qt-5/qvboxlayout.html
- QHBoxLayout : https://doc.qt.io/qt-5/qhboxlayout.html
- QFrame : https://doc.qt.io/qt-5/qframe.html
- QScrollBar : https://doc.qt.io/qt-5/qscrollbar.html
- QSplitter : https://doc.qt.io/qt-5/qsplitter.html

```

def generateSplitter(self):

    vbox = QVBoxLayout()
    hbox = QHBoxLayout()

    self.leftTopFrame = QFrame()
    self.leftTopFrame.setFrameShape(QFrame.StyledPanel)

    self.rightWidget = QWidget()
    self.rightWidget.resize(300, 300)
    self.layoutRight = QVBoxLayout()
    self.rightWidget.setLayout(self.layoutRight)

    self.scroll = QScrollArea()
    self.layoutRight.addWidget(self.scroll)
    self.scroll.setWidgetResizable(True)
    self.scrollContent = QWidget(self.scroll)

    self.scrollLayout = QVBoxLayout(self.scrollContent)
    self.scrollContent.setLayout(self.scrollLayout)
    self.scroll.setWidget(self.scrollContent)

    self.splitterLeft = QSplitter(Qt.Vertical)
    self.leftBottonFrame = QFrame()
    self.leftBottonFrame.setFrameShape(QFrame.StyledPanel)

    self.splitterLeft.addWidget(self.leftTopFrame)
    self.splitterLeft.addWidget(self.leftBottonFrame)
    self.splitterLeft.setSizes([VERTICAL_TOP_SIZE,VERTICAL_BOTTOM_SIZE])

    ''' Horizontal Splitter '''
    self.splitterCentral = QSplitter(Qt.Horizontal)
    self.splitterCentral.addWidget(self.splitterLeft)
    self.splitterCentral.addWidget(self.rightWidget)
    self.splitterCentral.setSizes([HORIZONTAL_LEFT_SIZE,HORIZONTAL_RIGHT_SIZE])
    hbox.addWidget(self.splitterCentral)

    vbox.addLayout(hbox)

    ''' Status Bar '''
    self.statusBar = QLabel()
    self.statusBar.setStyleSheet(STYLE_STATUSBAR)
    vbox.addWidget(self.statusBar)

    self.setLayout(vbox)
```

4.3 La frame inférieure gauche : la sélection d'un répertoire

Maintenant que nos frames sont définies, il s'agit de les remplir. Commençons par définir l'explorateur de disque local en bas à gauche. La sélection d'un répertoire entraîne la génération de la frame droite contenant les détails de chaque photo du répertoire. Pour réaliser cette frame on utilise l'objet PyQt QTreeView dans le cadre de l'implémentation du paradigme MVC, dans lequel l'instance de QTreeView constitue la Vue et une instance de l'objet QFileSystemModel constitue le Modèle. On filtre de manière à n'afficher que les répertoires (et pas les fichiers inclus).

```
def generateLeftBottomFrame(self):

    self.layoutLeftBottom = QVBoxLayout()

    '''Model for QTreeView'''
    self.modelTree = QFileSystemModel()
    self.modelTree.setRootPath(QDir.currentPath())
    self.modelTree.setFilter(QDir.Dirs) # Only directories

    ''' QTreeView '''
    self.treeLeftBottom = QTreeView()
    self.treeLeftBottom.setModel(self.modelTree)
    self.treeLeftBottom.setAnimated(False)
    self.treeLeftBottom.setIndentation(10)
    self.treeLeftBottom.setColumnWidth(0, 300)
    self.treeLeftBottom.expandAll()
    self.treeLeftBottom.selectionModel().selectionChanged.connect(self.onSelectFolder)
    self.layoutLeftBottom.addWidget(self.treeLeftBottom)
    self.leftBottonFrame.setLayout(self.layoutLeftBottom)
```

On voit que l'on associe l'évènement de changement de sélection à la méthode onSelectFolder décrite ci-après. Le travail de cette méthode consiste à parcourir le contenu du répertoire sélectionné.

```
def onSelectFolder(self, selected, deselected):

    try:
```

```
    currentIndex = selected.indexes()[0]
    self.currentDirectoryPath = self.modelTree.filePath(currentIndex)
    print(self.currentDirectoryPath)
    self.statusBar.setText("")
    self.exifImageCollection = []

    list_dir = os.listdir(self.currentDirectoryPath)
    files = [f for f in sorted(list_dir) if isfile(join(self.currentDirectoryPath, f))]
    for file in files:
        fullFilePath = os.path.join(self.currentDirectoryPath, file)
      if fullFilePath.endswith(".jpeg") or fullFilePath.endswith(".jpg") or fullFilePath.
endswith(".png"):

            currentExifImage  = EXIFImage()
            currentExifImage.fullFilePath = fullFilePath
            currentExifImage.filename = file
            tags = None

            try:
              ''' EXIF '''
              f_exif = open(fullFilePath, 'rb')
              tags = exifread.process_file(f_exif)
              #print(tags)
            except:
              print("A problem with EXIF data reading")

            ''' Location'''
            # 'GPS GPSLatitude', 'GPS GPSLongitude'] # [45, 49, 339/25] [4, 55, 716/25]
            # 'GPS GPSImgDirection' 'GPS GPSLatitudeRef'
            lat = ''
            long = ''
            heading = ''
            try:
              currentExifImage.lat, currentExifImage.long, currentExifImage.heading = get
_exif_location(tags)
            except:
              print("A problem with EXIF data reading")

            dt = None
            try:
              ''' Date Time '''
              dt = tags['EXIF DateTimeOriginal'] # 2021:01:13 14:48:44
            except:
              print("A problem with EXIF data reading")

            print (dt)
            dt = str(dt)
            indexSpace = dt.find(" ")
            currentExifImage.date = dt[0:indexSpace].replace(":", "-")
            currentExifImage.time = dt[indexSpace+1:]

            self.exifImageCollection.append(currentExifImage)
            print(currentExifImage)

    self.generateRightFrame()
```

Pour chaque image, on récupère les métadonnées incluses dans les données EXIF de chaque photo : latitude, longitude, direction DSC, date et heure. La fonction suivante extrait les trois métadonnées géographiques.

```
def get_exif_location(exif_data):
    """
    Returns the latitude and longitude, if available, from the provided exif_data
(obtained through get_exif_data above)
    """
    lat = None
    lon = None
    heading = 0

    gps_latitude = _get_if_exist(exif_data, 'GPS GPSLatitude')
    gps_latitude_ref = _get_if_exist(exif_data, 'GPS GPSLatitudeRef')
    gps_longitude = _get_if_exist(exif_data, 'GPS GPSLongitude')
    gps_longitude_ref = _get_if_exist(exif_data, 'GPS GPSLongitudeRef')
    gps_direction = _get_if_exist(exif_data, 'GPS GPSImgDirection')
    gps_direction_ref = _get_if_exist(exif_data, 'GPS GPSImgDirectionRef')

    if gps_latitude and gps_latitude_ref and gps_longitude and gps_longitude_ref:
        lat = _convert_to_degress(gps_latitude)
        if gps_latitude_ref.values[0] != 'N':
            lat = 0 - lat

        lon = _convert_to_degress(gps_longitude)
        if gps_longitude_ref.values[0] != 'E':
            lon = 0 - lon

    if str(gps_direction_ref) == "T": # Real North
        try:
            tab = str(gps_direction).split('/')
            print(tab)
            tab[0] = int(tab[0])
            tab[1] = int(tab[1])
            heading = tab[0]/tab[1]
        except:
            traceback.print_exc()

    return lat, lon, heading
```

4.4 La frame droite : les photographies à téléverser

Voyons à présent la génération de la frame de droite, c'est-à-dire la zone 3. C'est la fonction generateRightFrame qui a la charge de cette tâche. Ci-dessous la boucle incluse dans cette fonction qui génère dynamiquement chaque ensemble de composants graphiques pour chaque photo à téléverser.

On voit que l'on applique le même principe qu'entraperçu précédemment. On ajoute dynamiquement des widgets PyQt à des contrôles de disposition eux-mêmes ajoutés à la frame droite.

```
for currentExifImage in self.exifImageCollection:

    ''' Current image '''
    localWidget = ImageUpload()
    localLayout = QHBoxLayout()
    localLayout.setAlignment(Qt.AlignRight)
    localWidget.setLayout(localLayout)
    self.scrollLayout.addWidget(localWidget)
    self._currentUpload.append(localWidget)

    '''Local Left Widget'''
    localLeftWidget = QWidget()
    localLeftLayout = QFormLayout()
    localLeftLayout.setAlignment(Qt.AlignRight)
    localLeftWidget.setLayout(localLeftLayout)
```

```
localLayout.addWidget(localLeftWidget)

''' import? + Import Status '''
cbImport = QCheckBox("Import")
lblUploadResult = QLabel()
lblUploadResult.setStyleSheet(STYLE_IMPORT_STATUS)
localLeftLayout.addRow(cbImport, lblUploadResult)
localWidget.cbImport = cbImport
localWidget.lblUploadResult = lblUploadResult

''' File Name of picture '''
lblFileName = QLabel("Name: ")
lblFileName.setAlignment(Qt.AlignLeft)
lineEditFileName = QLineEdit()
lineEditFileName.setFixedWidth(WIDTH_WIDGET_RIGHT)
lineEditFileName.setText(currentExifImage.filename)
lineEditFileName.setAlignment(Qt.AlignLeft)
localLeftLayout.addRow(lblFileName, lineEditFileName)
localWidget.lineEditFileName = lineEditFileName

''' Shadow Real FileName '''
lblRealFileName = QLineEdit()
lblRealFileName.setText(currentExifImage.filename)
localWidget.lblRealFileName = lblRealFileName
localWidget.lblRealFileName.isVisible = False

''' Description '''
lblDescription = QLabel("Description: ")
lblDescription.setAlignment(Qt.AlignLeft)
lineEditDescription = QPlainTextEdit()
lineEditDescription.setFixedWidth(WIDTH_WIDGET_RIGHT)
localLeftLayout.addRow(lblDescription, lineEditDescription)
localWidget.lineEditDescription = lineEditDescription

''' Categories '''
lblCategories = QLabel("Categories: ")
lblCategories.setAlignment(Qt.AlignLeft)
lineEditCategories = QLineEdit()
lineEditCategories.setFixedWidth(WIDTH_WIDGET_RIGHT)
lineEditCategories.setAlignment(Qt.AlignLeft)
localLeftLayout.addRow(lblCategories, lineEditCategories)
localWidget.lineEditCategories = lineEditCategories

lblLocation = QLabel("Location: ")
lblLocation.setAlignment(Qt.AlignLeft)
lineEditLocation = QLineEdit()
lineEditLocation.setFixedWidth(WIDTH_WIDGET_RIGHT)
if currentExifImage.lat == None or currentExifImage.long == None:
    lineEditLocation.setText('')
else:
    lineEditLocation.setText(str(currentExifImage.lat) + '|' + str(currentExifImage.long)
+ "|heading:" + str(currentExifImage.heading))
lineEditLocation.setAlignment(Qt.AlignLeft)
localLeftLayout.addRow(lblLocation, lineEditLocation)
localWidget.lineEditLocation = lineEditLocation

lblDateTime = QLabel("Date Time: ")
lblDateTime.setAlignment(Qt.AlignLeft)
lineEditDateTime = QLineEdit()
lineEditDateTime.setFixedWidth(WIDTH_WIDGET_RIGHT)
lineEditDateTime.setText(currentExifImage.date + ' ' + currentExifImage.time)
lineEditDateTime.setAlignment(Qt.AlignLeft)
localLeftLayout.addRow(lblDateTime, lineEditDateTime)
localWidget.lineEditDateTime = lineEditDateTime

''' Image itself '''
label = QLabel()
pixmap = QPixmap(currentExifImage.fullFilePath)
pixmapResize = pixmap.scaledToWidth(IMAGE_DIMENSION, Qt.FastTransformation)
label.setPixmap(pixmapResize)
localLayout.addWidget(label)
localWidget.fullFilePath = currentExifImage.fullFilePath
```

4.5 La frame supérieure gauche : les données de l'import

Avant d'étudier l'import lui-même, voyons comment la frame supérieure gauche est construite. On procède comme précédemment : on alimente un contrôle de disposition en y ajoutant des widgets PyQt.

```
def generateLeftTopFrame(self):

    self.layoutLeftTop = QFormLayout()
    self.layoutLeftTop.setFormAlignment(Qt.AlignTop)

    self.lblUserName = QLabel("Username: ")
    self.lblUserName.setAlignment(Qt.AlignLeft)
    self.lineEditUserName = QLineEdit()
    self.lineEditUserName.setFixedWidth(WIDTH_WIDGET)
    self.lineEditUserName.setAlignment(Qt.AlignLeft)
    self.lineEditUserName.setText("")
    self.layoutLeftTop.addRow(self.lblUserName, self.lineEditUserName)

    self.lblPassword = QLabel("Password: ")
    self.lblPassword.setAlignment(Qt.AlignLeft)
    self.lineEditPassword = QLineEdit()
    self.lineEditPassword.setFixedWidth(WIDTH_WIDGET)
    self.lineEditPassword.setAlignment(Qt.AlignLeft)
    self.lineEditPassword.setEchoMode(QLineEdit.Password)
    self.layoutLeftTop.addRow(self.lblPassword, self.lineEditPassword)

    self.lblSource = QLabel("Source: ")
    self.lblSource.setAlignment(Qt.AlignLeft)
    self.lineEditSource = QLineEdit()
    self.lineEditSource.setFixedWidth(WIDTH_WIDGET)
    self.lineEditSource.setText("{{own}}")
    self.lineEditSource.setAlignment(Qt.AlignLeft)
    self.layoutLeftTop.addRow(self.lblSource, self.lineEditSource)

    self.lblAuthor = QLabel("Author: ")
    self.lblAuthor.setAlignment(Qt.AlignLeft)
    self.lineEditAuthor = QLineEdit()
    self.lineEditAuthor.setFixedWidth(WIDTH_WIDGET)
    self.lineEditAuthor.setText("")
    self.lineEditAuthor.setAlignment(Qt.AlignLeft)
    self.layoutLeftTop.addRow(self.lblAuthor, self.lineEditAuthor)

    self.lblCategories = QLabel("Categories: ")
    self.lblCategories.setAlignment(Qt.AlignLeft)
    self.lineEditCategories = QLineEdit()
    self.lineEditCategories.setText("")
    self.lineEditCategories.setFixedWidth(WIDTH_WIDGET)
```

```
self.lineEditCategories.setAlignment(Qt.AlignLeft)
self.layoutLeftTop.addRow(self.lblCategories, self.lineEditCategories)

self.lblLicense = QLabel("License: ")
self.lblLicense.setAlignment(Qt.AlignLeft)
self.lineEditLicense = QLineEdit()
self.lineEditLicense.setFixedWidth(WIDTH_WIDGET)
self.lineEditLicense.setText("{{self|cc-by-sa-4.0}}")
self.lineEditLicense.setAlignment(Qt.AlignLeft)
self.layoutLeftTop.addRow(self.lblLicense, self.lineEditLicense)

self.lblDescription = QLabel("Description: ")
self.lblDescription.setAlignment(Qt.AlignLeft)
self.lineEditDescription = QPlainTextEdit()
self.lineEditDescription.setFixedWidth(WIDTH_WIDGET)
self.layoutLeftTop.addRow(self.lblDescription, self.lineEditDescription)

separationLeftTopFrame = QLabel()
self.layoutLeftTop.addWidget(separationLeftTopFrame)

''' Button Import & None/All checkboxes'''
importWidget = QWidget()
importLayout = QHBoxLayout()
importWidget.setLayout(importLayout)

self.cbImportNone = QCheckBox("None")
self.cbImportNone.stateChanged.connect(self.cbImportNoneStateChanged)

self.cbImportAll = QCheckBox("All")
self.cbImportAll.stateChanged.connect(self.cbImportAllStateChanged)

self.btnImport = QPushButton("Import!")

self.btnImport.clicked.connect(self.onClickImport)

importLayout.addWidget(self.cbImportNone)
importLayout.addWidget(self.cbImportAll)
importLayout.addWidget(self.btnImport)
self.layoutLeftTop.addWidget(importWidget)
importWidget.setStyleSheet("border:1px solid #808080;");
self.cbImportNone.setStyleSheet("border:0px;");
self.cbImportAll.setStyleSheet("border:0px;");

self.btnImport.setStyleSheet(STYLE_IMPORT_BUTTON)

''' Layout Association of the Left Top Frame'''
self.leftTopFrame.setLayout(self.layoutLeftTop)
```

4.6 L'import des photos lui-même : les requêtes HTTP

La frame droite contient lors du lancement de l'import toutes les informations sur les photos à téléverser : leurs futurs noms, leurs emplacements sur le disque, les métadonnées qui vont servir à contextualiser les photos sur Wikimedia Commons. La procédure telle que décrite dans la documentation de l'API de Mediawiki est la suivante. Elle se déroule en quatre étapes :

1. Obtenir un token nécessaire pour se connecter avec un login et un mot de passe.
2. Se connecter grâce au login, au mot de passe et au token.
3. Puis pour chaque image à téléverser, on obtient un token CSRF.
4. Fort de ce token, on envoie une image vers Wikimedia Commons ainsi que ses différentes métadonnées mises en forme.

Nous pourrions utiliser la bibliothèque Pywikibot, qui permet d'utiliser facilement d'utiliser l'API Mediawiki en Python. Mais nous avons choisi de coder les quatre types d'appels en utilisant juste le module requests.

Documentation en ligne de l'upload de fichier *via* l'API Mediawiki : https://www.mediawiki.org/wiki/API:Upload

La bibliothèque Pywikibot : https://www.mediawiki.org/wiki/Manual:Pywikibot/fr

Le module requests : https://requests.readthedocs.io/en/master/

Voici le code de l'appel 1 (obtention du token).

```
self.S = requests.Session()

print(self.S)

# Step 1: Retrieve a login token
PARAMS_1 = {
  "action": "query",
  "meta": "tokens",
  "type": "login",
  "format": "json"
}
R = self.S.get(url=URL, params=PARAMS_1)
DATA = R.json()
print(DATA)

LOGIN_TOKEN = DATA["query"]["tokens"]["logintoken"]
print(LOGIN_TOKEN)
```

Puis celui de l'appel 2 (la connexion elle-même à l'aide du login et du mot de passe) :

```
# Step 2: Send a post request to login
PARAMS_2 = {
  'action': "clientlogin",
  'username': self.login,
  'password': self.password,
  'loginreturnurl': URL,
  'logintoken': LOGIN_TOKEN,
  'format': "json"
}

R = self.S.post(URL, data=PARAMS_2)
print(R.content)
print(R.json()['clientlogin']['status'])
if R.json()['clientlogin']['status'] != 'PASS':
  self.widget.statusBar.setText("Client login failed")
  return
```

Pour chaque image, on obtient un token CSRF (appel 3) :

```
# Step 3: Obtain a CSRF token
PARAMS_3 = {
  "action": "query",
  "meta":"tokens",
  "format":"json"
}
```

```
R = self.S.get(url=URL, params=PARAMS_3)
DATA = R.json()
print(DATA)

CSRF_TOKEN = DATA["query"]["tokens"]["csrftoken"]
print(CSRF_TOKEN)
```

Puis on envoie l'image elle-même et les données qui lui sont associées (appel 4) :

```
# Step 4: Post request to upload a file directly
PARAMS_4 = {
  "action": "upload",
  "filename": fileName,
  "format": "json",
  "token": CSRF_TOKEN,
  "ignorewarnings": 1,
  "comment": "PyCommonist upload: " + fileName,
  "text": text
}

try:
  # Here test if the file is still there...
  FILE = {'file':(fileName, open(FILE_PATH, 'rb'), 'multipart/form-data')}

  R = self.S.post(URL, files=FILE, data=PARAMS_4)
  print(R)
  resultUploadImage = R.json()['upload']['result']
  print(resultUploadImage)
  element.lblUploadResult.setText(resultUploadImage)
  self.widget.alreadyUploaded = self.widget.alreadyUploaded + 1
  self.widget.statusBar.setText("..." + str(self.widget.alreadyUploaded) + "/" + str
(self.widget.numberImagesChecked) + " image(s) are successfully uploaded")
except:
  traceback.print_exc()
  element.lblUploadResult.setText("FAILED")
  self.widget.alreadyUploaded = self.widget.alreadyUploaded + 1
  self.widget.statusBar.setText("..." + str(self.widget.alreadyUploaded) + "/" + str
(self.widget.numberImagesChecked) + " image(s) are successfully uploaded")
```

4.7 L'import des photos lui-même : le multithreading

Un problème persiste toutefois : si on fait ces différents téléversements dans le thread courant, celui de l'application PyCommonist, celle-ci va "freezer". Elle sera en effet inutilisable le temps de l'import multiple. D'où la nécessité de faire chaque import dans un thread dédié. Plusieurs options s'offrent à nous : on aurait pu ainsi utiliser un pool de threads grâce à l'objet QThreadPool. On peut également utiliser la classe QThread, solution adoptée ici.

- Documentation de QThreadPool : https://doc.qt.io/qt-5/qthreadpool.html
- QThread : https://doc.qt.io/qt-5/qthread.html

L'idée est d'associer un thread à chaque image à importer. On lance le premier import et donc le premier thread. Quand ce premier import est terminé, on interrompt ce premier thread et on donne la main au second import. On lance le second thread qui lui est associé. Quand ce second import est terminé, on interrompt le thread associé et on donne la main au troisième import, etc. La boucle de déclaration d'un thread par image à téléverser :

```
for element in self.widget._currentUpload:
  if element.cbImport.isChecked():
    print("for element in self.widget._currentUpload:")
    path = self.widget.currentDirectoryPath
    session = self.S
    index = self.widget.currentImageIndex

    thread = QThread()
    self.widget.threads.append(thread)
    process = ProcessImageUpload(element, self.widget, path, session, index)
    self.widget.workers.append(process)
    self.widget.workers[index].moveToThread(self.widget.threads[index])
    self.widget.threads[index].started.connect(self.widget.workers[index].process)
    self.widget.currentImageIndex = self.widget.currentImageIndex + 1
```

On lance alors le premier import :

```
self.widget.threads[0].start()
```

Lorsqu'un import est terminé, il lance le suivant. Si c'est le dernier import et qu'il est terminé, on détruit cette collection de threads utilisés pour les imports. Entre chaque import, on temporise quelques secondes ; pour cela on utilise la classe QTimer.

```
def runNextThread(self):
  if self.index < self.widget.numberImagesChecked - 1:
    print("Start next process")
    timer = QTimer()
    timer.setInterval(6); # To avoid a 502 error (first test ok with 10)
    timer.start()
    self.widget.threads[self.index + 1].start()
  elif self.index == self.widget.numberImagesChecked - 1:
    print("Call Clean threads")
    self.widget.cleanThreads()
```

Documentation de QTimer : https://doc.qt.io/qt-5/qtimer.html

5. Conclusion

Plusieurs points restent ouverts pour améliorer PyCommonist. Notamment le temps d'affichage des miniatures des images dans la frame droite. L'extrait du code suivant est celui qui permet de créer les vignettes pour chaque image du répertoire. Une solution serait peut-être de générer les vignettes à la volée au gré du défilement de la scrollbar verticale.

```
label = QLabel()
pixmap = QPixmap(currentExifImage.fullFilePath)
pixmapResize = pixmap.scaledToWidth(IMAGE_DIMENSION, Qt.FastTransformation)
label.setPixmap(pixmapResize)
```

Une autre amélioration est relative au déploiement et au fait de fournir un package d'installation sur macOS, Linux et Windows grâce à l'utilisation de PyInstaller ou de cx_Freeze.
Documentation de PyInstaller : https://www.pyinstaller.org/
cx_Freeze : https://cx-freeze.readthedocs.io/
Le code de PyCommonist est disponible sur Github à cette adresse : https://github.com/benprieur/PyCommonist
Les images téléversées sur Wikimedia Commons grâce à PyCommonist sont catégorisées sur Wikimedia Commons : https://commons.wikimedia.org/wiki/Category:Uploaded_with_PyCommonist

Deep Learning : programmation d'un réseau à convolution avec TensorFlow/Keras

Dans le n°244, nous avions introduit le Deep Learning avec la programmation d'un réseau de neurones de type perceptron multicouche pour reconnaître des chiffres manuscrits. Pour poursuivre la découverte de l'univers de l'intelligence artificielle, nous décrirons aujourd'hui les réseaux à convolution qui ont démontré ces dernières années des performances remarquables en analyse d'images. L'exemple détaillé montre comment programmer un tel réseau, avec les outils TensorFlow/Keras [1][2], pour distinguer la présence d'un chat ou d'un chien sur des photos.

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 [5]. Je suis enthousiasmé par les progrès en apprentissage machine, d'autant qu'Internet rend accessibles toutes les ressources (tutoriels, outils…) pour mettre en œuvre les techniques les plus récentes, comme celle du « Deep Learning », ou « apprentissage profond » en français.

Limites à l'utilisation d'un perceptron multicouche

Le plus simple pour analyser une image avec un réseau de neurones est de présenter en entrée du réseau la totalité des pixels de l'image. Avec un perceptron multicouche (MLP : « *Multi-Layer Perceptron* »), cette approche fonctionne bien avec des visuels de petite taille (cf. exemple détaillé dans le N°244), mais est difficilement possible pour des tailles plus importantes comme des photos. Par exemple, une définition de 256 x 256 pixels utilise environ 65 000 valeurs pour une représentation en niveaux de gris et trois fois plus, soit près de 200000 valeurs, pour un codage RGB couleurs. Puisqu'avec un réseau MLP, le nombre de neurones doit être en cohérence avec le nombre d'entrées, traiter une telle image nécessitera des couches avec des dizaines de milliers de neurones. Le nombre de poids à calculer pour les connexions peut alors vite dépasser le milliard, ce qui n'est pas compatible avec les puissances disponibles actuellement pour les algorithmes d'apprentissage (**cf. encadré n°1**).

Pour exploiter des images telles que des photos, il est donc nécessaire de réduire leur taille, tout en conservant les informations indispensables à la compréhension de leur contenu. Une première approche peut consister à développer des algorithmes de traitement pour extraire des éléments représentatifs afin de les utiliser en entrées d'un perceptron. Cette solution présente l'inconvénient d'avoir à développer un algorithme spécifique pour chaque type d'application : la détection de la position des yeux, de la bouche et du nez pour faire de la reconnaissance de visage, l'analyse de la couleur et de la texture de la peau pour identifier des fruits, la détection de contours pour l'identification de piétons dans une scène routière, etc.

Rupture avec les réseaux à convolution

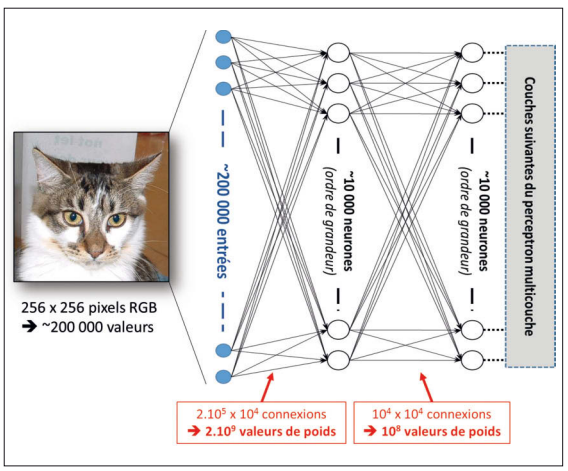
Une autre solution consiste à identifier des motifs caractéristiques sur des images par apprentissage. Ces éléments, appelés « features », doivent permettre, en détectant leur présence ou leur absence, d'identifier un contenu à analyser. **L'encadré n°2** illustre le principe de l'utilisation de tels motifs pour discriminer la présence d'une croix ou d'un rond sur une image. Dans l'exemple, pour faciliter la compréhension, les motifs utilisés correspondent à des zones directement visibles, mais ce n'est pas toujours le cas ; les « features » peuvent également être obtenues après traitements sur des valeurs de pixels. Les réseaux de neurones à convolution (CNN : « *Convolutional Neural Networks* ») permettent, grâce à des couches appelées « features extraction », de définir à partir d'une base d'images, des motifs caractéristiques à utiliser pour leur interprétation (classification, localisation d'éléments…).

Ces réseaux ont démontré leur puissance depuis l'édition 2012 du challenge ILSVRC (ImageNet Large Scale Visual Recognition Challenge) [3]. Cette compétition annuelle regroupe les meilleures équipes de recherche mondiales (publiques et privées), autour du thème de l'analyse d'images. L'objectif est d'exploiter une base d'environ 1 200 000 photos organisée en 1000 catégories (chats, chiens, maisons, voitures, camions…), pour réussir à trier 100 000 nouvelles images. Les résultats sont classés en fonction du taux de

Encadré n°1

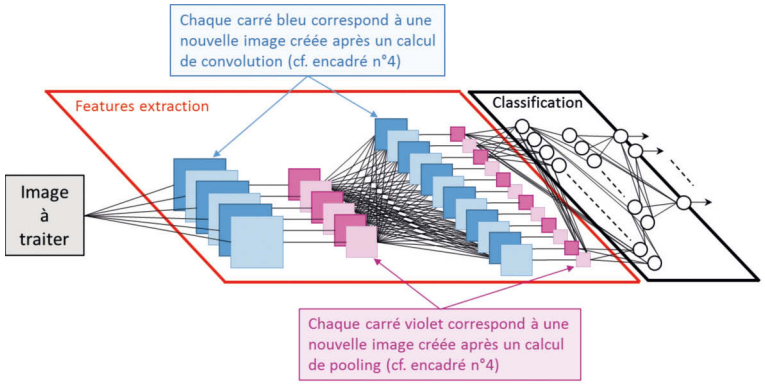
Limitation des perceptrons multicouche

Le nombre de paramètres d'un perceptron multicouche augmente extrêmement rapidement avec la taille des images à exploiter en entrée du réseau. Comme illustré sur le visuel, pour une photo couleur de 256x256 pixels, le nombre de valeurs à fixer pour les poids peut facilement atteindre le milliard, ce qui dépasse largement les capacités d'apprentissage avec les puissances de calculs actuelles. Il est donc nécessaire de réduire la taille des images pour pouvoir les exploiter avec un réseau de type perceptron.



Encadré n°3

Architecture d'un réseau à convolution



Un réseau de neurones à convolution se décompose en 2 parties :

1. **« Extraction de features »** : détermination des motifs caractéristiques avec des calculs de filtrage par convolution et pooling (cf. encadré n°4)
2. **« Classification/Décision »** : exploitation des motifs définis à l'étape précédente pour calculer la sortie du réseau (souvent une classification)

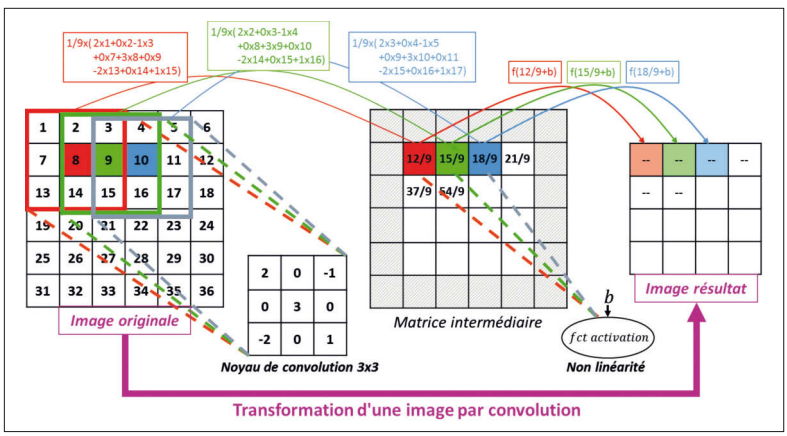
Les différentes architectures de réseaux à convolution diffèrent par le nombre de blocs de convolution/pooling et par les caractéristiques de chaque bloc (taille des noyaux de convolution, logique de pooling,…)

Encadré n°4

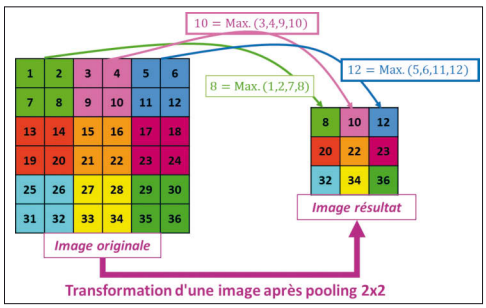
Calculs de convolution et de pooling

Convolution : l'objectif est de faire ressortir des éléments sur une image, avec un calcul en 2 étapes :

1. Application d'un noyau de convolution (différentes tailles possibles : 3x3, 5x5,…) pour modifier chaque pixel en exploitant les valeurs des pixels périphériques (combinaison linéaire)
2. Utilisation d'une fonction d'activation neuronale (par ex. Relu) pour calculer la valeur finale pour chaque pixel (introduction d'un non-linéarité).



Ce calcul se généralise pour obtenir un résultat à partir de plusieurs images, chacune ayant son propre noyau de convolution. Le fonctionnement est identique avec simplement l'ajout d'une étape pour additionner les matrices intermédiaires calculées pour chaque image avant l'application de la non-linéarité.



Pooling : l'objectif est de réduire la taille d'une image ; une solution couramment utilisée consiste à remplacer, un ensemble de pixels par celui avec la valeur la plus élevée (la zone pour le pooling peut prendre différentes tailles : 2x2, 3x3…).

réponses correctes avec l'utilisation de 2 critères : proposition de 5 catégories par image (« Top-5 accuracy »), ou d'une seule (« Top-1 acuracy »). En 2012, l'équipe de l'université de Toronto a remporté la compétition en utilisant pour la 1re fois un réseau à convolution baptisé AlexNet [7].

Cette victoire a déclenché un développement important des réseaux à convolution, avec une amélioration régulière des performances grâce aux évolutions des architectures de réseaux et à l'augmentation des puissances de calcul [4]. Du point de vue historique, il est intéressant de noter que le principe des réseaux à convolution avait été proposé dès la fin des années 1990, par le français Yann Le Cun [6]. C'est d'ailleurs pour ces travaux sur l'apprentissage profond qu'il a reçu en 2018 le prix Turing, considéré comme l'équivalent du prix Nobel en informatique.

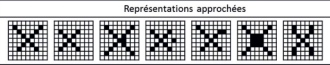
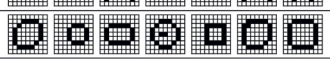
Structure d'un réseau à convolution

Comme détaillé dans **l'encadré n°3,** un réseau à convolution se compose d'une 1re série de couches pour identifier les motifs caractéristiques de l'image, suivie de neurones interconnectés formant un perceptron multicouche pour calculer les sorties. La taille du réseau doit être définie selon le problème à traiter. Le nombre de couches de convolution et de pooling (**Encadré n°4**) est à adapter en fonction de la taille des images présentées en entrée. L'objectif est de réussir à obtenir

Encadré n°2

Principe d'utilisation de motifs

Supposons que nous voulions discriminer des images avec des ronds et des croix. La classification doit fonctionner pour les représentations de référence, mais aussi pour des représentations approchées, comme détaillée dans le tableau ci-dessous :

	Référence	Représentations approchées
Croix		
Rond		

La détection des 4 motifs suivants  permet de discriminer les images avec la logique suivante :

SI 2<=  <=4 ET SI 2<=  <=4 ET SI  =0 ET SI  =0 → **"X"**

SI 0<=  <=2 ET SI 0<=  <=2 ET SI 2<=  <=4 ET SI 2<=  <=4 → **"O"**

Exemples:				
			Aucun	Aucun
			Aucun	Aucun
				
	Aucun			

Il est important de noter que ces motifs ne sont pas liés à une position, mais qu'ils peuvent être détectés n'importe où dans l'image. Cette solution rend robuste la caractérisation du contenu, puisque la croix ou le rond peuvent se situer à n'importe quel endroit.

après la dernière couche de pooling des motifs de taille réduite, typiquement avec quelques pixels de côté. Les réseaux les plus simples alternent les couches de convolution et de pooling, mais ce n'est pas une obligation. Il est possible de réaliser plusieurs convolutions successives dans le but de calculer des motifs plus sophistiqués.

Le nombre de neurones en sortie du réseau dépend du résultat attendu : pour un problème de classification selon n catégories, on choisira en général n sorties qui correspondent à la probabilité d'appartenance à chaque classe ; pour un tri binaire, une seule sortie avec une valeur réelle entre 0 et 1 peut convenir en interprétant le résultat comme une attirance vers l'une ou l'autre des conclusions. Le perceptron de classification comporte généralement peu de couches (typiquement entre 1 et 3) avec un nombre de neurones par couche à adapter en fonction du nombre de motifs présentés en entrée.

Mise en œuvre d'un réseau à convolution

Un réseau à convolution s'utilise de la même manière qu'un perceptron multicouche :

- Analyse du problème pour définir la taille des images à utiliser en entrées du réseau et les résultats attendus en sorties,
- Identification d'une base d'exemples pour l'apprentissage, c'est-à-dire d'un ensemble de couples composés d'images avec les sorties souhaitées. Généralement cette base est divisée en deux groupes : les exemples effectivement utilisés pour l'apprentissage et ceux employés pour tester les performances du réseau.
- Choix de l'architecture du réseau avec l'organisation des étapes de convolution/pooling et la définition de la taille du perceptron pour la classification.
- Phase d'apprentissage (également appelée entraînement) pour calculer à partir des exemples, les valeurs des noyaux de convolution, et celles des poids/biais des couches de classification.
- Phase de test pour évaluer les performances du réseau sur d'autres exemples que ceux utilisés pour l'entraînement. Si les résultats sont satisfaisants, le réseau pourra être exploité avec d'autres données. Dans le cas contraire, il faut revenir aux étapes précédentes pour modifier, soit la base d'apprentissage, soit l'architecture du réseau.

Après la théorie, passons à la pratique sur un exemple de classification de photos de chats et de chiens.

Choix d'un jeu de données

Pour développer une application avec un réseau à convolution, il est nécessaire de disposer d'une base d'exemples d'images étiquetées, c'est-à-dire pour lesquelles les conclusions souhaitées du réseau sont connues. Kaggle (www.kaggle.com) est une plateforme de partage dans le domaine de l'analyse de données et du machine learning, qui met à disposition de très nombreux jeux de données (« *datasets* »). Pour notre exemple, nous avons sélectionné une base avec au total 10 000 photos de chats et de chiens. **L'encadré n°5** détaille comment télécharger ces images et les stocker sur disque afin de pouvoir les exploiter avec le programme. Il est à noter que Kaggle propose d'autres bases d'images de nos animaux favoris, certaines comportant jusqu'à 25 000 photos. Le choix de notre dataset est justifié par la recherche de durées de calculs « raisonnables » pour l'apprentissage et par une structuration des fichiers directement exploitable avec les outils TensorFlow/Keras.

Programme Python avec TensorFlow/Keras

L'utilisation des bibliothèques TensorFlow/Keras est possible dans tous les environnements de développement Python (Windows, macOS ou Linux), sous réserve d'installer le module TensorFlow. Pour expérimenter le programme de cet article, nous conseillons deux approches :

- Installer un environnement Python, par exemple celui du site www.anaconda.com (distribution "Individual Édition" gratuite), et travailler en local sur son ordinateur,
- Ou se connecter à « Colab » (https://colab.research.google.com/notebooks/intro.ipynb), avec un compte Google ; l'avantage de cette solution est de ne nécessiter aucune installation locale puisque tout fonctionne sur le « cloud » via l'utilisation d'un navigateur Internet.

Le programme débute par l'importation des modules :

```
# Importation des modules
import tensorflow.keras.preprocessing.image as img
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import os
```

`'tensorflow'` constitue la brique essentielle pour construire et exploiter des réseaux de neurones. `'tensorflow.keras.preprocessing.image'` est un module qui permet d'exploiter très facilement des images dans les phases d'apprentissage et de test. `'matplotlib.pyplot'` propose des fonctions graphiques utilisées pour afficher des images lors de l'analyse des résultats. `'numpy'` et `'os'` sont utiles pour manipuler des arborescences de fichiers et réaliser des calculs sur des tableaux.

Il est ensuite nécessaire de préciser les répertoires où sont stockées les images pour l'apprentissage et les tests. Dans le cas d'un fonctionnement avec Colab, il sera sans doute

Encadré n°5

Utilisation d'un dataset avec Kaggle

Procédure à suivre

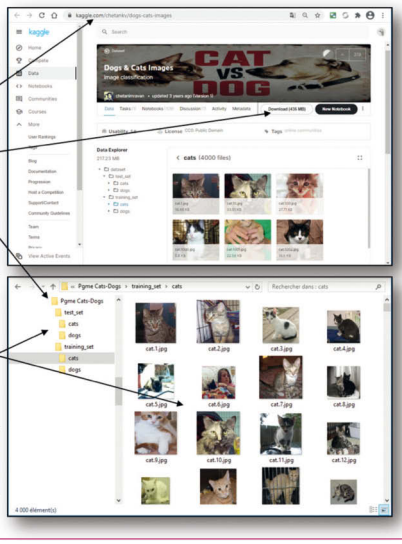
1/ se connecter à l'adresse:
https://www.kaggle.com/chetankv/dogs-cats-images

2/ Télécharger le fichier "archive.zip" (435 Mo)

3/ Extraire du fichier "archive.zip" uniquement les répertoires "training_set" et "test_set" situés dans le répertoire "dataset" et les stocker sur disque dans un répertoire "Pgme Cats-Dogs"

4/ Résultat:
- Le répertoire "test_set" comporte 2 répertoires "cats" & "dogs", avec chacun 1000 images
- Le répertoire "training_set" comporte 2 répertoires "cats" & "dogs" avec chacun 4000 images

5/ Le fichier source du programme python est à stocker dans le répertoire "Pgme Cats-Dogs" (afin de pouvoir exploiter les répertoires avec les images)



nécessaire de déplacer les fichiers pour qu'ils soient accessibles depuis le cloud.

```
# Références pour les fichiers images de 'training' et de 'validation'
path_train ='./training_set/'
path_valid ='./test_set/'
```

La suite du code correspond à la description du réseau à convolution utilisé. Grâce aux objets disponibles dans le module `'keras'` il suffit d'énumérer la composition les couches successives. Comme illustré dans l'encadré n°6, nous avons choisi d'utiliser un réseau classique. La partie « features extraction » est composée de 5 phases de convolution/pooling avec à chaque étape un doublement du nombre d'images générées par convolution (dans le même temps leur taille est à chaque fois réduite de moitié). Ce choix est arbitraire, mais l'expérience montre que ce type d'architecture fonctionne généralement assez bien.

`'tf.keras.layers.Flatten()'` ne constitue pas une couche du réseau, mais permet de « mettre à plat » les pixels des motifs résultats de la partie « features extraction », pour les utiliser comme entrées pour la classification. Celle-ci utilise un perceptron avec 3 couches, composées respectivement de 256, 128 et 64 neurones. En sortie le réseau comporte un unique neurone activé par une fonction sigmoïde. Cette solution est souvent utilisée pour les classifications à 2 états, car la valeur de la sigmoïde entre 0 et 1 peut s'interpréter comme la probabilité de l'une ou l'autre classe.

```
# Construction d'un réseau à convolution
MonReseau = tf.keras.models.Sequential([
    # 1ère couche de convolution/pooling avec en entrée des images RGB
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(200,200,3)),
    tf.keras.layers.MaxPooling2D(2,2),
    # 2ème couche de convolution/pooling
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # 3ème couche de convolution/pooling
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # 4ème couche de convolution/pooling
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # 5ème couche de convolution/pooling
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(512, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Perceptron de classification avec 3 couches
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.25),
    # 1 seul neurone de sortie
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Des couches `'tf.keras.layers.Dropout(0.25)'` sont utilisées pour améliorer les performances du réseau. Il ne s'agit pas réellement de couches supplémentaires, mais simplement de l'ajout d'un traitement particulier dans le fonctionnement de la couche suivante, dans le but d'éviter le phénomène de sur-apprentissage (« *over-fitting* ») Ce dernier se produit lorsque le réseau est « réglé » trop précisément avec les données d'apprentissage, mais qu'il ne donne pas des résultats satisfaisants lors de la généralisation sur les données de test. Le fonctionnement du « dropout » consiste à forcer aléatoirement (ici avec un taux de 25%) des entrées de neurones à zéro dans le but de mieux répartir les calculs entre les différents éléments d'une même couche [8].

Pour vérifier la définition du réseau, l'appel à la fonction `'.summary()'` permet d'afficher la description détaillée. Elle comporte notamment le nombre et la taille des images après chaque étape de convolution et de pooling (**Encadré n°6**), ainsi que le nombre de paramètres (noyau de convolution, poids des connexions et biais des neurones). Au total ce réseau comporte 3 707 201 paramètres qui seront à calculer par l'algorithme d'apprentissage pour obtenir le fonctionnement souhaité sur les images d'entraînement.

```
# Affichage de la description du réseau
MonReseau.summary()
```

La suite du programme détaille les données à utiliser pour l'apprentissage et les tests. La bibliothèque `'keras.preprocessing.image'` permet d'exploiter directement des images stockées sur disque dur. Les instances de la classe `'ImageDataGenerator'` spécifient les traitements à appliquer aux images. Dans notre programme nous utilisons uniquement le paramètre `'rescale'` pour transformer les valeurs entières des pixels en réels sur l'intervalle [0,1]. Cette classe propose également d'autres fonctionnalités, non détaillées dans cet article, pour enrichir une base d'images par des opérations de symétrie, décalage, rotation…

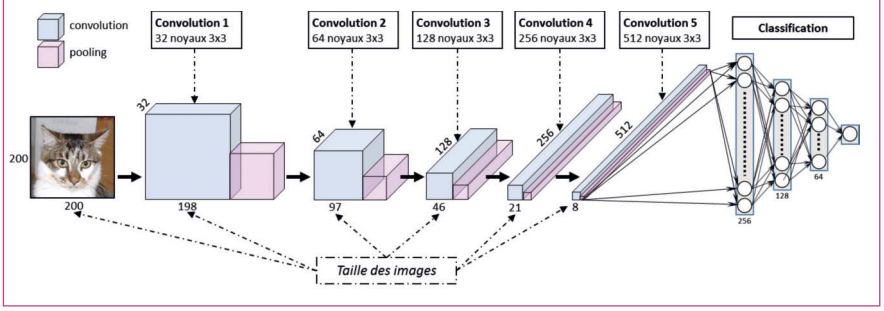
Les objets `'generator_train'` et `'generator_valid'` génèrent à partir des fichiers, les couples (image, conclusion) utilisés pour les calculs d'apprentissage et de test. Le paramètre `'target_size=(200,200)'` impose l'utilisation d'images de 200x200 pixels ; étant donné que les images disponibles sont de dimensions très variées, leurs tailles sont modifiées par contraction ou étirement dans les 2 dimensions pour correspondre à la cible. Il en résulte des déformations plus ou moins importantes, mais qui restent compatibles avec la logique de fonctionnement du réseau. Le paramètre `'class_mode='binary''` indique l'utilisation d'une conclusion binaire.

```
# Instanciation de la classe 'ImageDataGenerator' avec le paramètre
# 'rescale' pour transformer les valeurs de pixels en réels sur [0,1]
datagen_train = img.ImageDataGenerator(rescale=1/255)
datagen_valid = img.ImageDataGenerator(rescale=1/255)

# Objet pour générer des données (images et labels) pour le 'training':
# - les images sont automatiquement chargées à partir du répertoire
#   'directory' en adaptant leur taille aux dimensions 'target_size'
# - les labels sont construits à partir des noms des répertoires
# (la fonction 'flow_from_directory' retourne un objet 'iterator')
generator_train = datagen_train.flow_from_directory(
    directory=path_train,     # répertoire où sont stockées les images
    target_size=(200,200),    # résolution des images chargées en mémoire
    class_mode='binary')      # classement en 2 catégories

# Objet pour générer des données (images et labels) pour le 'test':
# - les images sont automatiquement chargées à partir du répertoire
#   'directory' en adaptant leur taille aux dimensions 'target_size'
# - les labels sont construits à partir des noms des répertoires
generator_valid = datagen_valid.flow_from_directory(
    directory=path_valid,     # répertoire où sont stockées les images
    target_size=(200,200),    # résolution des images chargées en mémoire
    class_mode='binary')      # classement en 2 catégories
```

Le programme se termine par la phase d'apprentissage et de validation. L'entraînement fonctionne pendant 15 cycles, en exploitant les images produites par `'generator_train'`. L'utilisation de l'algorithme `'RMSprop'` avec un taux d'apprentissage fixé à 0,001 permet d'obtenir des performances satisfaisantes puisque, sur les images de validation, le taux de classifications correctes atteint 85% (**Encadré n°7**).

Encadré n°6

Architecture du réseau à convolution utilisé



```
# Définition des paramètres pour l'apprentissage
MonReseau.compile(loss='binary_crossentropy',
                  optimizer=tf.keras.optimizers.RMSprop(lr=0.001),
                  metrics=['accuracy'])

# Apprentissage avec les données construites par 'generator_train'
hist=MonReseau.fit(x=generator_train, epochs=15, validation_data=generator_valid)
```

Le programme propose deux fonctions complémentaires pour analyser les performances du réseau. La fonction `'AfficherCourbes'` représente graphiquement les courbes d'évolution de l'erreur résiduelle et du taux de réussite durant l'entraînement pour les données d'apprentissage et de test. Ces courbes permettent d'observer d'une part si la progression de l'apprentissage est régulière et d'autre part si les résultats sont cohérents entre les données d'apprentissage et de test.

```
#-------------------------------------------------------------------
# GRAPHIQUE pour analyser L'évolution de l'apprentissage
#  => courbes erreurs / fiabilité au cours des cycles d'apprentissage
#-------------------------------------------------------------------
def AfficherCourbes(hist):
  # création de la figure ('figsize' pour indiquer la taille)
  plt.figure(figsize=(8,8))
  # evolution du pourcentage des bonnes classifications
  plt.subplot(2,1,1)
  plt.plot(hist.history['accuracy'],'o-')
  plt.plot(hist.history['val_accuracy'],'x-')
  plt.title("Taux d'exactitude des prévisions",fontsize=15)
  plt.ylabel('Taux exactitude',fontsize=12)
  plt.xlabel("Itérations d'apprentissage",fontsize=15)
  plt.legend(['apprentissage','validation'], loc='lower right',fontsize=12)
  # évolution des valeurs de L'erreur résiduelle moyenne
  plt.subplot(2,1,2)
  plt.plot(hist.history['loss'],'o-')
  plt.plot(hist.history['val_loss'],'x-')
  plt.title('Erreur résiduelle moyenne',fontsize=15)
  plt.ylabel('Erreur',fontsize=12)
  plt.xlabel("Itérations d'apprentissage",fontsize=15)
  plt.legend(['apprentissage','validation'], loc='upper right',fontsize=12)
  # espacement entre les 2 figures
  plt.tight_layout(h_pad=2.5)
  plt.show()

# Affichage des courbes de convergence d'apprentissage et de validation
AfficherCourbes(hist)
```

La fonction `'AfficherImages'` sélectionne des images dans les bases d'apprentissage ou de validation et les affiche en comparant la conclusion du réseau avec la classification attendue. Une analyse visuelle des images mal classées peut permettre de juger si l'erreur du réseau provient d'un fonctionnement non satisfaisant ou si elle peut s'expliquer par la mauvaise qualité de l'image.

```
# Affichage de 7x4=28 images avec les résultats de classification
def AfficherImages(path_dir,IndiceClasse,iDebut):
  ListeFichiers = os.listdir(path_dir)    # liste des fichiers du répertoire
  plt.figure(figsize=(12.5,7.6), dpi=100) # taille (en inch) de la figure
  for NoImg in range(7*4):
    # chargement de l'image au format PIL
    Img_PIL=img.load_img(path_dir+ListeFichiers[iDebut+NoImg],target_size=(200,200))
    # transformation au format ndarray avec normalisation sur [0,1]
    Img_Array = img.img_to_array(Img_PIL)/255
    # transformation en une liste (format pour 'predict') avec une seule image
    Img_List = np.expand_dims(Img_Array,axis=0)
    # affichage de l'image
    plt.subplot(4,7,NoImg+1) # pour 'subplot' les indices commencent à 1
    plt.imshow(Img_PIL)      # pour afficher une image au format PIL
    plt.xticks(ticks=[])     # suppression des graduations en x
    plt.yticks(ticks=[])     # suppression des graduations en y
    # calcul de la conclusion de réseau et comparaison avec celle attendue
    if round(MonReseau.predict(Img_List)[0][0]) == IndiceClasse:
      plt.title('Bien classé',pad=1,size=10,color='Green')
    else:
      plt.title('Mal classé',pad=1,size=10,color='Red')
  plt.show()

# Affichages des résultats sur les images de validation
# L'attribut 'class_indices' (dict) associe les indices aux noms de classes
AfficherImages(path_valid+'cats/',generator_train.class_indices.get('cats'),0)
AfficherImages(path_valid+'dogs/',generator_train.class_indices.get('dogs'),0)
```

Conclusion

L'application détaillée dans cet article montre la puissance des réseaux à convolution : grâce aux outils TensorFlow/Keras, il a été possible avec quelques dizaines de lignes de programme Python de développer, sur un simple PC, une application pour distinguer des photos de chat ou de chien avec une fiabilité de reconnaissance honorable. Il y a encore quelques années, atteindre de telles performances n'aurait pas été imaginable ! Pourtant nous n'avons que commencé à entrouvrir la boite de Pandore, car d'autres réseaux à convolution, plus évolués, permettent d'atteindre des taux de reconnaissance proches des 100%. Aussi nous invitons les lecteurs curieux à poursuivre cette découverte sur Internet ou à attendre de prochains numéros de Programmez ! pour continuer à explorer le monde fabuleux de l'intelligence artificielle…

Références :

Documentations en ligne :

[1] TensorFlow : https://www.tensorflow.org/

[2] Bibliothèque Keras : https://www.tensorflow.org/api_docs/python/tf/keras

[3] Challenge ILSVRC (ImageNet) : http://www.image-net.org/challenges/LSVRC/

[4] Classification des performances des réseaux avec la base ImageNet (Challenge ILSVRC) https://paperswithcode.com/sota/image-classification-on-imagenet

[5] Site de l'école d'ingénierie informatique EPSI (membre de HEP Éducation) https://www.epsi.fr/

Publications historiques :

[6] 1998 "Gradient-based learning applied to document recognition" Yann LeCun, Léon Bottou, Yoshua Bengio, & Patrick Haffner (réseau LeNet-5) http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

[7] 2012 "ImageNet Classification with Deep Convolutional" Alex Krizhevsky, Ilya Sutskever & Geoffrey E. Hinton (réseau AlexNet) http://www.cs.toronto.edu/~hinton/absps/imagenet.pdf

[8] 2014 "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" Nitish Srivastava Geoffrey Hinton Alex Krizhevsky & Co. https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

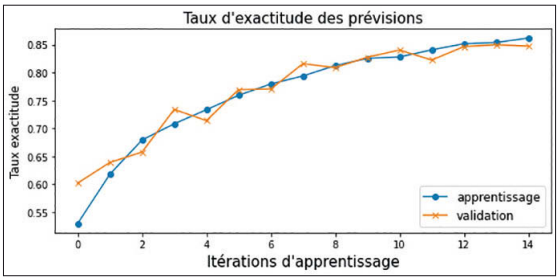
Ouvrages pour approfondir :

« *Quand la machine apprend* », Yann Le Cun, Editions Odile Jacob, 2019

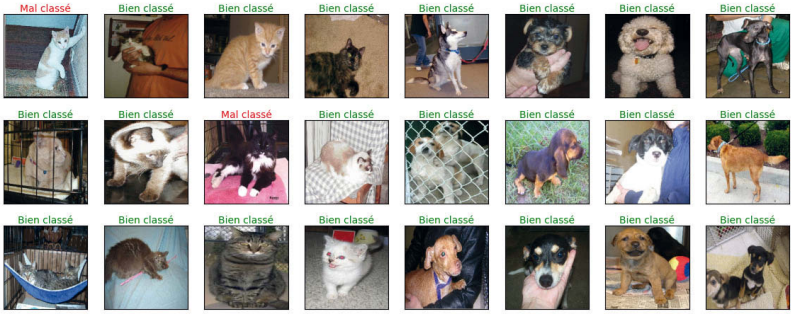
« *L'apprentissage profond avec Python* », François Chollet, Editions machinelearning.fr, 2020

Encadré n°7

Performances de reconnaissance



À chaque itération de l'entraînement, le programme analyse le taux de reconnaissances correctes sur les 8000 images de la base d'apprentissage, et sur les 2000 images de la base de test. Les courbes montrent des progrès réguliers avec une croissance similaire des 2 courbes. Le résultat final avec environ 85 % de bonnes détections constitue un résultat honorable.



Exemples d'images de la base de test avec les résultats du réseau : les 2 photos de chat mal classées, donc reconnues comme des chiens, peuvent s'expliquer par le fait que dans la 1re image le chat n'est pas dans une position habituelle (donc non apprise) et que sur la 2e image, la race du chat à poils longs, peut être confondue avec un chien.

Les anciens numéros de PROGRAMMEZ!

Le magazine des développeurs

226

236

238

239

240

241

HS 01 été 2020

242

HS 02

Tarif unitaire 6,5 € (frais postaux inclus)

TECHNOSAURES

Le magazine à remonter le temps !

N°1

N°2

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

N°3

N°4 Standard **10 €**
N°4 Deluxe **15 €**

N°5

N°6

☐ 226	: __ ex	☐ 241	: __ ex
☐ 236	: __ ex	☐ HS 01	: __ ex
☐ 238	: __ ex	☐ 242	: __ ex
☐ 239	: __ ex	☐ HS 02	: __ ex
☐ 240	: __ ex		

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 __ exemplaires x 6,50 € = __ € ..... __ € 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 246

Marwa THLITHI
Ingénieur R&D chez INVIVOO

Regex : pourquoi il faut aimer les expressions régulières ?

PARTIE 1

Utilisée à la manière des outils de recherche de texte dans un document, une expression régulière, aussi dénommée « **regex** » ou « **regexp** », fournit un moyen concis et flexible pour la correspondance de chaînes de texte, tel que des caractères particuliers, mots ou motifs de caractères.

Écrites dans un langage formel, langage de programmation avec symboles, qui peut être interprété par un processeur d'expression régulière, elles sont des expressions « Joker » intelligentes pour rechercher, isoler ou remplacer des chaînes de texte.

Dans ce dossier, je vais aborder plusieurs points :

- Module regex en Python
- Recherche et export des données
- Regroupement et groupes nommés
- Quantificateurs
- Compilation des regex
- Caractères et séquences d'échappement
- Barre oblique inversée

Module regex en Python

Python fournit un module complet permettant l'utilisation des expressions régulières qui est le module « **re** ». Les fonctions clés de ce module sont :

- La fonction **re.search()** qui permet de voir si une chaîne correspond à une expression régulière, semblable à l'utilisation de la méthode **find()** pour les chaînes de caractères,
- La fonction **re.findall()** qui permet d'extraire des fragments d'une chaîne de caractère correspondant à votre expression régulière, semblable à une combinaison de **find()** et coupe avec les crochets. Une coupe avec les crochets appliquée sur une chaîne de caractères permet d'avoir une sous-chaîne de la chaîne principale. Prenons comme exemple la chaine 'Magazine Programmez', si on veut extraire uniquement le nom du magazine, on peut utiliser l'instruction suivante : chaine[9:] qui va retourner la sous-chaine 'Programmez'.
- La fonction **re.compile()** qui permet de compiler une expression régulière et de la réutiliser sans la recréer.

Avant de commencer les exemples d'utilisation des regex, je vous propose ce guide rapide d'expressions régulières. Ce guide contient les « caractères marqueurs » de base utilisés pour les regex ainsi que leurs traductions. **(voir tableau ci-dessous)**.

Caractère	Traduction en langage regex
^	**début** d'une ligne
$	**fin** d'une ligne
.	Corresponds **à n'importe** quel caractère
\s	Corresponds à un **espace**
\S	Corresponds à n'importe quel caractère **sauf l'espace**
*	**Répète** un caractère **zéro** ou **plusieurs fois**
+	**Répète** un caractère **une** ou **plusieurs fois**
[aeiou]	Corresponds à un **caractère unique** de la liste des caractères indiqués
[^XYZ]	Corresponds à un caractère unique **qui n'est pas** dans la liste XYZ
[a-z0-9]	Lettres minuscules de **A à Z** ou chiffres de **0 à 9**
\w	Corresponds à n'importe quel caractère **alphanumérique** ([a-zA-Z0-9_])
\W	Corresponds à n'importe quel caractère **non alphanumérique** ([^a-zA-Z0-9_])
\d	Corresponds à n'importe quel caractère **numérique**
\D	Corresponds à n'importe quel caractère **non numérique**
(	Indique le **début de l'extraction** de la chaîne
)	Indique la **fin de l'extraction** de la chaîne
R|S	L'expression régulière R **ou** S

Recherche des données

Les expressions régulières permettent de faire de la recherche de données. En Python, c'est la fonction **re.search()** qui permet de faire ce traitement en renvoyant un objet contenant un tuple (*span*) correspondant à la position du début et la position de fin de la sous-chaîne et la sous-chaîne correspondante à la regex. Si la regex ne correspond à aucune sous-chaîne de la chaîne de caractères, un objet None sera retourné. L'utilisation de cette fonction dans une condition renvoie Vrai/Faux selon si la chaîne correspond à la regex.

Si nous disposons ici de cette chaîne de caractères :

```
chaine = 'From: Using the : character'
```

Voici un exemple d'utilisation de la fonction **re.search()** qui recherche le mot « Using » dans la chaîne ci-dessus :

```
import re
re.search('Using', chaine)
```

Le résultat de cette recherche est l'objet suivant :

```
<_sre.SRE_Match object; span=(6, 11), match='Using'>
```

Présentons maintenant deux comparaisons entre la fonction **re.search()** et les fonctions utilisées pour la recherche des données dans les chaînes de caractères **find()** et **startswith()**. On veut afficher la chaîne si elle contient le mot « From » avec la fonction **find()**, qui renvoie un nombre désignant la position du mot dans la chaîne:

```
if chaine.find('From') >= 0:
  print(chaine)
```

la version avec les regex :

```
import re

if re.search('From', chaine):
```

```
print(chaine)
```

Maintenant, si on veut vérifier que la chaîne commence par le mot « From », on peut utiliser la fonction **startswith()** pour les chaînes de caractères :

```
if chaine.startswith('From'):
  print(chaine)
```

En utilisant le module « **re** », nous pouvons perfectionner la recherche précédente en ajoutant le caractère spécial « ^ » afin de vérifier que la chaîne commence par le mot « From » :

```
import re

if re.search('^From', chaine):
  print(chaine)
```

Dans cet exemple, l'utilisation de la fonction **startswith()** est plus efficace que l'utilisation d'une expression régulière pour vérifier qu'une chaîne de caractères commence par un tel motif. En effet, la création d'une expression régulière est associée à la création d'une machine à états, ce qui possède un coût. L'utilisation des regex est très utile, mais quand il s'agit de vérifier uniquement qu'une chaîne de caractère commence ou se termine par un motif, l'utilisation des fonctions **startswith()** et **endswith()** est recommandée, car elle est moins coûteuse que la création d'une machine à états.
Il existe aussi une autre fonction de recherche dans le module « **re** » : la fonction **re.match()**. À la différence de la fonction **re.search()** qui analyse la chaîne à la recherche d'une position où la regex correspond, **re.match()** détermine si la regex correspond dès le début de la chaîne. Si la regex ne correspond pas dès le début de la chaîne, un objet None sera retourné. L'exécution de ce code :

```
import re
re.match('From', chaine)
```

renvoie l'objet suivant :

```
<_sre.SRE_Match object; span=(0, 4), match='From'>
```

Il s'agit du même type d'objet renvoyé par la fonction **re.search()**. Si nous exécutons maintenant ce code :

```
import re
print(re.match('Using', chaine))
```

Le résultat retourné est **None**, car la regex ne correspond pas dès le début de la chaîne. Ce n'est pas le résultat trouvé lorsque nous avons fait cette recherche avec la fonction **re.search()**, car cette dernière analyse toute la chaîne.

Export des données

Si nous voulons réellement extraire les chaînes correspondantes à notre expression régulière, nous utilisons la fonction **re.findall()**. Le code suivant montre l'extraction de tous les nombres de la chaîne de caractères **x** :

```
import re

x = 'My 2 favorite numbers are 19 and 42'
y = re.findall('[0-9]+',x)
print(y)
```

L'expression régulière utilisée :

```
[0-9]+      Un ou plusieurs chiffres
```

Ce code retourne une liste de plusieurs chaînes secondaires qui correspondent à l'expression régulière, c'est-à-dire tous les chiffres de la chaîne **x** :

```
['2', '19', '42']
```

Si aucune chaîne secondaire de **x** ne correspond à l'expression régulière, une liste vide sera retournée. Cela est le cas de l'exemple suivant :

```
y = re.findall('[AEIOU]+',x)
print(y)
```

```
[AEIOU]+  Un ou plusieurs caractères de la liste des caractères [AEIOU]
```

Comme il s'agit d'une correspondance sensible à la casse, c'est-à-dire qu'il y a une différence entre les lettres majuscules et minuscules, aucun caractère de la chaîne **x** ne correspond à aucun caractère de la liste **[AEIOU]**. D'où ce résultat :**[]**

Le caractère Joker

Le point « . » est un caractère Joker qui correspond à tout caractère. Si vous ajoutez le symbole astérisque « * », le caractère sera répété « n'importe quel nombre de fois ».

Quantificateurs

Les quantificateurs permettent de répéter un ensemble de caractères plusieurs fois. Deux types de quantificateurs existent :

- Les quantificateurs explicites dans leur forme qui indiquent le nombre de répétitions et qui peuvent s'écrire de 4 façons :

Quantificateurs	Signification
{n}	Répétition **n fois** consécutives
{n,m}	Répétition de **n** à **m fois** consécutives
{,n}	Répétition de **zéro** à **n fois** consécutives
{n,}	Répétition au moins **n fois** consécutives

- Les quantificateurs préconçus utilisant des métacaractères et qui peuvent être associés à des recherches voraces ou non-voraces. Les types de recherche vorace et non-vorace sont définis et détaillés dans la partie suivante. Voici la liste du deuxième type de quantificateurs :

Quantificateurs	Signification
?	Répète un caractère **zéro** ou **une fois**
*	Répète un caractère **zéro** ou **plusieurs fois**
+	Répète un caractère **une** ou **plusieurs fois**
*?	Répète un caractère **zéro** ou **plusieurs fois non voraces**
+?	Répète un caractère **une** ou **plusieurs fois non-vorace**

Voici un exemple de vérification d'un numéro de téléphone qui utilise ces deux types de quantificateurs :

```
re.match('^0[0-9]([.\s]?[0-9]{2}){4}', '06 98 65 22 76')
```

Dans la regex de cet exemple, nous avons utilisé le quantificateur « ? » pour indiquer la répétition de l'un des caractères « . » ou espace 0 ou 1 seule fois. Nous avons indiqué le nombre 4 de répétitions de 2 chiffres précédés d'un espace ou

d'un point ou de rien. Donc, l'exemple du numéro de téléphone mentionné correspond bien à la regex, d'où le résultat :

```
<_sre.SRE_Match object; span=(0, 14), match='06 98 65 22 76'>
```

Les numéros de téléphone 06.88.56.12.66 et 0789443899 correspondent aussi à la regex utilisée.

Recherche vorace

Les caractères « * » et « + » répétés déplacent vers l'extérieur dans les deux sens (vorace) afin de correspondre à la plus grande chaîne possible. L'exemple suivant illustre ce type de recherche :

```
chaine = 'From: Using the : character'
y = re.findall('^F.+:', chaine)
print(y)
```

Dans cet exemple, nous faisons une extraction de tout le texte qui précède le caractère « : » en utilisant l'expression régulière suivante :

^ F.+:	Chaîne commençant par la lettre « F » suivie par au moins un caractère et finissant par le caractère « : »

Comme nous avons deux fois le caractère « : » dans la chaîne et comme il s'agit d'une recherche vorace, la recherche s'étend jusqu'au dernier caractère « : » de la chaîne tant que la regex est vérifiée. D'où ce résultat :

```
['From: Using the :']
```

Recherche non vorace

Les regex qui répètent du code ne sont pas toutes voraces! Si vous ajoutez le caractère « **?** », les caractères « + » et « * » se détendent un peu...
Pour illustrer ce type de recherche, reprenons l'exemple ci-dessus pour la recherche vorace et adaptons-le afin d'extraire uniquement le texte précédant la première apparition du caractère « : ».

```
chaine = 'From: Using the : character'
y = re.findall('^F.+?:', chaine)
print(y)
```

Nous avons rajouté le caractère « ? » après le « .+ » afin de s'arrêter à la première apparition du caractère « : »

^ F.+ ?:	Chaîne commençant par la lettre « F » suivie par au moins un caractère, mais non vorace et finissant par le caractère « : »

D'où le résultat suivant :

```
['From:']
```

Amélioration de l'extraction de données

Nous avons vu plus haut, comment extraire des données avec les expressions régulières. Nous pouvons affiner notre recherche avec **re.findall()** et déterminer séparément quelle portion de la chaîne de caractères est à extraire en utilisant des **parenthèses** dans la regex. Les parenthèses ne font pas partie de la recherche, mais elles précisent où **commence** et où **se termine** la chaîne à extraire.

Voici un exemple qui consiste à extraire une adresse mail (adresse mail de l'expéditeur) d'une chaîne de caractère.
La chaîne que nous allons traiter :

```
chr = 'From marwa.thlithi@invivoo.com Sat Jan 5 09:14:16 20'
```

Le code permettant d'extraire l'adresse mail :

```
y = re.findall('\S+@\S+',chr)
print(y)
```

Cette expression régulière permet d'extraire l'adresse mail :

\S+@\S+	Au moins un caractère sans espace avant et après le caractère « @ »

Le résultat est :

```
['marwa.thlithi@invivoo.com']
```

Imaginons maintenant que nous avons une autre adresse mail dans notre chaîne de caractères (adresse mail du destinataire) et que nous voulons extraire uniquement l'adresse mail de l'expéditeur.
Voici la nouvelle chaîne que nous allons traiter :

```
chr = 'From marwa.thlithi@invivoo.com Sat Jan 5 09:14:16 20 to admin@invivoo.com'
```

Si nous appliquons le code et la regex précédents, nous obtenons le résultat suivant :

```
['marwa.thlithi@invivoo.com','admin@invivoo.com']
```

Cela ne correspond pas au résultat attendu. Afin de pouvoir extraire uniquement l'adresse mail de l'expéditeur, il faut affiner notre recherche et être plus précis dans notre extraction. Pour cela, nous pouvons utiliser cette proposition d'expression régulière :

^ From (\S+@\S+)	Chaîne commençant par le mot « From » suivi par un espace. Ensuite, nous commençons l'extraction en utilisant le caractère « (» de tout mot constitué du caractère « @ » précédé et suivi d'au moins un caractère sans espace. La fin de l'extraction est indiquée par le caractère «) »

Nous pouvons utiliser aussi l'expression régulière suivante :

^ From.*? (\S+@\S+)	Chaîne commençant par le mot « From » suivi par n'importe quel nombre de caractères, mais non vorace. Ensuite, nous commençons l'extraction en utilisant le caractère « (» de tout mot constitué du caractère« @ » précédé et suivi d'au moins un caractère sans espace. La fin de l'extraction est indiquée par le caractère «) »

NB : *Dans la deuxième proposition de regex, il faut faire attention à ne pas oublier le caractère « ? », car si on l'oublie, on aura une recherche vorace qui ne donnera pas le résultat attendu qui est l'adresse mail de l'expéditeur, mais plutôt la dernière adresse mail dans la chaîne de caractère.*

La suite dans le n°247

Le CADEAU idéal pour toutes/tous les geeks !

UNE HISTOIRE DE LA MICRO-INFORMATIQUE

Volume 3 : 90 nouvelles machines, + d'ordinateurs français !

116 pages. Format mook.

16€

UNE HISTOIRE DE LA MICRO-INFORMATIQUE

Volume 2 : les systèmes d'exploitation de 1956 à 2007

100 pages. Format mook.

16€

Commandez directement sur
www.programmez.com/catalogue/livres

Migrer ses applications Symfony 4 en Symfony 5

Abdel-Latif Mabrouck Matroud

Ingénieur développement full-stack Web, Osaxis

Laura Fontaine

Consultante AMOA - Développeuse full-stack Web, Osaxis

Exemples de codes : symfony.com

Abdel-Latif Mabrouck M.

Les développeurs utilisent de plus en plus de frameworks. Ils offrent, entre autres, une meilleure structuration de code pour le développement des applications. C'est ainsi qu'en 2017, le framework Symfony reposant sur le langage PHP, développé par Sensiolabs, atteignait le milliard de téléchargements. Deux ans plus tard, en novembre 2019, sort la 5e version du framework et donc le même enthousiasme mêlé d'inquiétude pour tous les développeurs l'utilisant : quelles sont les nouveautés ? Dois-je franchir le pas pour cette version et si oui, comment migrer mon application réalisée dans une version antérieure ?

Symfony est un framework open source PHP. Il est basé sur une architecture MVC (Modèle-Vue-Contrôleur). Il se définit comme un ensemble de composants PHP modulables et adaptables qui, en facilitant le développement, permet un gain de temps considérable dans la création et surtout la maintenance d'une application Web. Il est aussi possible de l'associer à d'autres frameworks tel que Bootstrap pour la partie design des pages ou encore des frameworks JavaScript tels Angular ou React pour des traitements côté client.

Installation

Pour son fonctionnement, Symfony a besoin d'un environnement composé de :

- PHP pour interpréter le code PHP
- d'un serveur Apache ou Nginx pour le serveur HTTP
- d'une base de données (MySql, SQLServer, PostgreSQL, Oracle…) pour les données

Symfony 4 : les grands principes

Avec sa 4e version, Symfony propose deux solutions logicielles :

- un squelette (skeleton) simple comprenant uniquement les packages de base
- un squelette pour site internet (website skeleton) qui intègre en plus de ceux de base, toute une catégorie de packages nécessaires à une application Web classique

On favorisera la première solution, plus légère, pour la création d'un micro-service ou d'une API par exemple.

L'une des nouveautés phares à relever dans cette version 4 est le plug-in Symfony Flex, automatisant les tâches les plus courantes comme l'action et la configuration de nouveaux packages. Bien que concrètement disponible depuis la version 3.3 de Symfony, ce plug-in n'a été adopté massivement que depuis la version 4 qui l'intègre dans ses bonnes pratiques. Si l'on compare à la version 3, nous étions obligés d'éditer le fichier AppKernel.php, le fichier config.yml et le fichier routing.yml pour la prise en compte de changements liés à l'ajout ou la suppression d'un package. Un seul fichier à mettre à jour au lieu de trois, on ne peut qu'apprécier !

Une autre spécificité de la version 4 de Symfony consiste à indiquer à son application d'utiliser le meilleur algorithme de hachage possible disponible sur le serveur. Pour ce faire, il faut tout d'abord modifier le fichier security.yaml.

CODE 1

```
# config/packages/security.yaml
security:
  # ...
  encoders:
    App\Entity\User:
      algorithm: auto
      cost: 14
```

En positionnant la variable « algorithm » à « auto », l'application passe en algorithme automatique **(CODE 1)**. Il suffit ensuite d'ajouter à votre repository UserRepository.php une nouvelle fonction comprenant la string à hacher **(CODE 2)**.

CODE 2

```
// src/Repository/UserRepository.php
namespace App\Repository;

// ...
use Symfony\Component\Security\Core\User\PasswordUpgraderInterface;

class UserRepository extends EntityRepository implements PasswordUpgrader
Interface
{
  // ...

  public function upgradePassword(UserInterface $user, string $new
EncodedPassword): void
  {
    // this code is only an example; the exact code will depend on
    // your own application needs
    $user->setPassword($newEncodedPassword);
    $this->getEntityManager()->flush($user);
  }
}
```

Cette fonction vérifiera à chaque connexion si un meilleur hachage est disponible, et si tel est le cas, il sera mis à jour. Passons à Symfony 5 !

Symfony 5 : les grands principes

Symfony 5 présente les mêmes caractéristiques que la version 4. Sur la forme par contre, cette version apporte deux composants représentatifs de sa maturité :

STRING : string est un composant orienté objet qui permet de travailler avec des chaînes de caractère en UTF-8. Il dispose de différentes fonctions comme le « replace » qui permet de remplacer un contenu dans une chaîne de caractères **(CODE 3)**, d'autres fonctions qui permettent de transformer la chaîne de caractères **(CODE 4)** ou encore la fonction « endsWith » qui peut être utilisée dans le cas où on veut valider la terminaison d'une sous chaîne dans une chaîne **(CODE 5)**.

CODE 3

```
$text =u('This is a déjà-vu situation.')
  ->trimEnd('.')
  ->replace('déjà-vu', 'jamais-vu')
  ->append('!');
// $text = 'This is a jamais-vu situation!'
```

CODE 4

```
<?php

namespace App\Controller;

// ...
use function Symfony\Component\String\u;

/**
 * @Route("/book")
 */
class BookController extends AbstractController
{
  // ...

  /**
   * @Route("/new", name="book_new", methods={"GET","POST"})
   * @param Request $request
   * @return Response
   * @throws ORMException
   * @throws OptimisticLockException
   */
  public function new(Request $request): Response
  {
    $book = new Book();
    $form = $this->createForm(BookType::class, $book);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
      try {
        if (!$this->bookService->exist($book)) {
          if ($book->getTitle() !== NULL) {
            $lengthTitle = u($book->getTitle())->length();
            $book->setTitle(
              u($book->getTitle())
                ->upper()
                ->padStart($lengthTitle + 2, '- ')
                ->padEnd($lengthTitle + 4, ' -')
            );
          }
          $this->bookService->save($book);

          $this->addFlash(
            'success',
            'Le livre ' . $book->getTitle() . ' a bien été créé.'
          );

          return $this->redirectToRoute('book_index');
        } else {
          $this->addFlash(
            'danger',
            'Un livre le même titre et auteur existe déjà.'
          );
        }
      } catch (DBALException $e) {
        $this->addFlash(
          'danger',
          'Une erreur s\'est produite lors de l\'ajout du livre. Veillez réessayer plustard.'
        );
      }
    }

    return $this->render('book/new.html.twig', [
      'book' => $book,
      'form' => $form->createView(),
    ]);
  }
}
```

CODE 5

```
// using Symfony's String
if (u($theString)->endsWith('.html')) {
  // ...
}
```

NOTIFIER : notifier est un composant permettant de gérer plus simplement l'envoi de notifications sur différents types de canaux, tels SMS, mail, Chat, etc. Par exemple, l'utilisateur peut être averti à chaque connexion si l'adresse IP utilisée est nouvelle :

```
<?php
namespace App\EventListener;

// ...
use Symfony\Component\Notifier\Message\SmsMessage;

class UserLocaleSubscriber implements EventSubscriber
{
  /** @var TexterInterface */
  private $notifier;
  /** @var UserService */
  protected $userService;

  public function __construct(TexterInterface $notifier, UserService $user
Service)
  {
    // ...
  }

  public function getSubscribedEvents()
  {
```

```
    // ...
  }

  public function onSecurityInteractiveLogin(InteractiveLoginEvent $event)
  {
    /** @var User $user */
    $user = $event->getAuthenticationToken()->getUser();
    $ipAddresse = $event->getRequest()->getClientIp();
    if (!empty($user->getIpAddresses())) {
      if (!in_array($ipAddresse, $user->getIpAddresses(), true)) {

        $user->addIpAddresse($ipAddresse);

        $sms = new SmsMessage(
          $user->getPhoneNumber(),
          $this->userService->getMessage($ipAddresse)
        );

        $this->notifier->send($sms);
      }
    }
  }
}
```

En plus de ces quelques ajouts, on y gagnerait aussi niveau performance. En effet, selon les équipes ayant implémenté Symfony 5, ce dernier comporterait 37 000 lignes de code de moins que la version précédente, engendrant des performances accrues par rapport à la version 4 (temps de réponse moindre lors de l'exécution d'applications reposant sur Symfony).

La migration de Symfony 4 vers Symfony 5

Comme on a pu le constater, la version 5 de Symfony ne révolutionne pas la version 4. Pour cette raison, le portage d'une application implémentée en version 4 vers la version 5 se fait sans trop de douleur. Celle-ci peut en effet bien se passer si vous respectez ces différentes étapes :

1. Vérifiez que les configurations serveur soient compatibles avec la nouvelle version
2. Si vous êtes en version mineure inférieure à 4.3, passez par la version 4.4. Elle contient la plupart des fonctionnalités de la version 5 et vous remontera tous les warnings (via developer toolbar).
3. Vérifiez les compatibilités des packages utilisés et corrigez au préalable les évènements du framework qui ont changé de nom entre les deux versions.

Les nouveautés de Symfony 5.1

En plus des modules présentés précédemment, les principaux ajouts de la 5.1 sont :

- Ajout du composant Uid appelé aussi Uuid, généralement utilisé pour sécuriser les urls à la place des ids (souvent déclarées en auto_increment).
- Ajout mineur d'un composant validateur permettant de tester les noms de domaine
- Notre coup de cœur : possibilité de priorisation des routes grâce au composant annotation. Plus la valeur de notre route est élevée, plus elle est prioritaire : ainsi la requête de la valeur la plus élevée est privilégiée.

Par exemple, si on se referre à la version 4, dans le cas où on a les deux routes suivantes "/users/export" et "/users/{username}", on était obligé de mettre la route "/users/export" au-dessus de "/users/{username}" pour prioriser la route "/users/export". Alors qu'avec cette nouvelle version, le simple fait de rajouter l'attribut « priority » dans les annotations de la route priorise la route de l'export à celle qui retourne les informations concernant un utilisateur :

```
<?php

namespace App\Controller;

use App\Entity\User;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

/**
 * @Route("/user")
 */
class UserController extends AbstractController
{
  /**
   * @Route("/{username}", name="user_show", methods={"GET"})
   * @param User $user
   * @return Response
   */
  public function show(User $user): Response
  {
    // ...
  }

  /**
   * @Route("/{export}", name="user_export", priority="1" methods
={"GET"})
   * @return Response
   */
  public function export(): Response
  {
    // ...
  }

  // ...
}
```

Conclusion

La version 5 de Symfony n'étant qu'une évolution naturelle de la version 4.4, il n'existe pas de différences notables comme chacun a pu le constater entre la v3 et la v4 (changements d'architecture, notions de Bundles, etc.). Plus généralement, il semble désormais peu probable que chaque future version de Symfony remette considérablement en question la version majeure précédente, Symfony ayant désormais atteint un bon degré de maturité. Souhaitons que la version 6, attendue pour fin 2021, soit conforme à nos prévisions, facilitant ainsi le portage dans la nouvelle mouture de la solution tout en profitant des dernières fonctionnalités implémentées.

L'art de faire du refactoring

Le refactoring peut s'apparenter à l'entretien d'une maison. Faire le ménage régulièrement permet d'éviter de perdre de précieux jours… A moins que vous ne préfériez tout repeindre une fois par an ?

Jules PERRODON

Consultant App & Infrastructure chez Exakis-Nelite

Lead-dev sur une solution d'IoT grande échelle

Bruno SOLFOROSI

Référent technique chez Exakis-Nelite

Architecte sur des projets de développement Azure

Bref, le refactoring c'est la pratique de restructurer et modifier du code ou une architecture existante sans impacter son comportement. Cette pratique apporte plusieurs avantages que l'on abordera dans cet article.

Nous verrons d'abord ce qu'est-ce et quand le mettre en place. Puis nous verrons les bonnes pratiques de développement applicables ainsi que les méthodologies possibles.

Il est légitime de se poser la question "*pourquoi modifier quelque chose qui fonctionne ?*". Le refactoring demande à la personne qui l'exécute de prendre du recul et de se poser de nouvelles questions. Il est possible de parler de refactoring de code, mais il s'agit surtout d'une mentalité applicable sur tous les domaines. Il est possible d'appliquer les principes de cet article sur des architectures logicielles ou cloud, de la documentation, et même sur vos habitudes de vie.

Pourquoi et par qui ?

Le but premier, que vous devez toujours avoir en tête, est la production d'un code ou d'une architecture lisible et maintenable. C'est un exercice difficile et le cheminement mental permettant d'arriver à une réponse aux besoins métier ne permet pas toujours d'obtenir une solution élégante. Il faut voir le développement d'une fonctionnalité comme un cycle itératif, une fois le besoin compris vient la première implémentation. Elle ne peut être parfaite, mais doit répondre au besoin. Ensuite, libre à vous de peaufiner votre création encore et encore.

Le refactoring doit vous permettre d'obtenir un code plus lisible et maintenable. Il n'est pas directement question d'optimisation de coûts ou de performances. C'est pour vous l'occasion de découvrir de nouvelles méthodes, outils ou librairies et de vous améliorer. **Figure 1**

Il n'est pas nécessaire d'être le créateur original de la fonction soumise au refactoring, l'important est la compréhension du besoin auquel elle répond. N'oubliez pas : vous devez modifier l'implémentation et non le comportement ! Le refactoring n'est pas forcément compliqué, mais il demande une profonde réflexion de design.

Quand faire du refactoring ?

Vous avez développé une nouvelle fonctionnalité. Cette dernière répond parfaitement aux besoins métier. Est-il maintenant temps de transformer votre nouveau code en commit git et l'oublier à jamais ? Avec le refactoring, pas vraiment ! Vous êtes dans l'état parfait pour commencer la réécriture de votre code.

Est-ce que votre solution sera facilement modifiable pour, à terme, répondre à de nouveaux besoins métier ? Un autre développeur pourra-t-il reprendre votre travail rapidement ?

Autre cas : une nouvelle fonctionnalité doit être ajoutée dans une solution existante, avant de commencer l'implémentation, il est intéressant de faire du refactoring pour faciliter l'intégration de cette nouvelle fonctionnalité. En fin de compte, vous allez gagner du temps !

Sur quoi faire du refactoring ?

On associe souvent le refactoring au code informatique, car il existe une infinité d'algorithmes permettant de résoudre un même problème. Un développeur travaillant sur l'implémentation ou la correction d'une fonctionnalité se posera de nombreuses questions :

- Mon code est-il assez propre ?
- Est-il facile à comprendre ?
- Pourra-t-on ajouter d'autres fonctionnalités ?
- Devra-t-on bientôt changer d'outils de développement ?

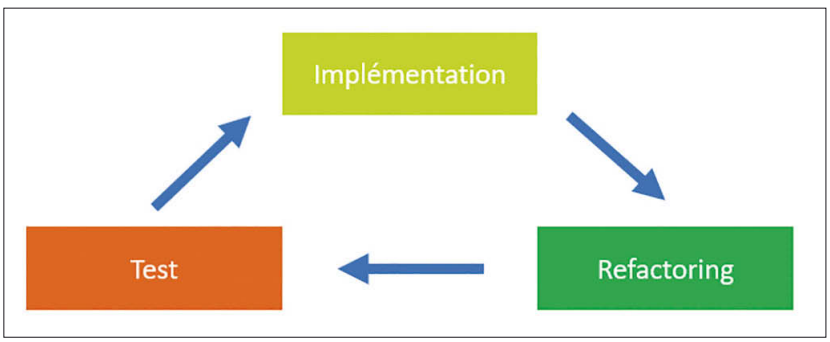
Il existe pourtant de nombreux domaines pour lesquels il est possible de se poser ces questions, parfois sans même s'en rendre compte :

- La maison est-elle bien rangée ?
- Changer une ampoule me demandera-t-il beaucoup d'effort ?
- Devrais-je refaire toute l'électricité pour brancher mon mixeur ?
- Pourrais-je encore trouver des dosettes de café compatibles avec ma cafetière dans 5 ans ?

Lorsque l'on crée ou maintient un produit, prendre du recul et réfléchir à ces points peut impliquer une remise en question sur de nombreux domaines :

- L'architecture générale : c'est souvent un cas extrême, car il implique de modifier des piliers sur lesquels repose le produit,
- Les outils et matériaux utilisés : changer d'outil est important pour rester performant, mais cela implique parfois de changer de matériaux (en informatique, on parlera de langages, de frameworks, etc.),
- La documentation : Il faut parfois prendre encore un peu de recul et penser à tous ceux qui devront utiliser le produit, ainsi qu'aux collègues qui auront la charge de le maintenir,
- L'intégration : souvent coûteux lorsqu'elles deviennent com-

Figure 1



plexes. L'intégration d'un produit dans un environnement peut-être coûteux en temps et source d'erreurs. Revoir ce processus peut être profitable,

- La qualité : selon les normes en vigueur et les retours utilisateurs sur un produit, faire évoluer le processus qualité au fur et à mesure est un point crucial,
- Le monitoring : pouvoir suivre de manière macro et micro l'utilisation du produit est important. Ne pas oublier de faire évoluer les outils de monitoring. Cela implique parfois de modifier le produit lui-même,
- La complexité : plus un produit est complexe à comprendre, plus il sera difficile à maintenir. Dans du code informatique, on essaie parfois de minimiser le nombre de lignes et de compresser certaines expressions. Et pourtant, cela entraîne régulièrement l'augmentation de la complexité des fonctions et ainsi la difficulté à reprendre et maintenir le code. Il est parfois conseillé de préférer l'utilisation d'algorithmes plus simples.

Les points cités peuvent être non exhaustifs selon le type de produit, mais restent des points clés dans un projet informatique. Il est important des garder à l'esprit que tous les constituants du cycle de vie peuvent être remis en question et retravaillés.

Les bonnes questions à se poser

Prenons un cas concret d'un projet informatique : vous travaillez sur une suite de plusieurs logiciels dont le socle (l'architecture de base) est identique. Cependant, ils évoluent de manière séparée, et de nouveaux sont créés régulièrement tandis que d'autres, bien que toujours utilisés, n'évoluent plus depuis de nombreux mois.

Au fil du temps, vous vous rendez compte que le socle utilisé est améliorable, et que certains bugs peuvent être corrigés. Vous avez listé toutes ces améliorations / corrections et attendez le GO du responsable produit pour commencer le travail. Celui-ci revient vers vous et après vous avoir complimenté pour votre travail et votre implication (car c'est un bon responsable qui ne veut surtout pas vous démotiver), vous donne un ensemble de questions auxquelles il vous demande de répondre :

• Quel est le temps nécessaire ?

Il s'agit là de la première question à se poser. Estimer et réestimer une tâche est une bataille de tous les jours dans le monde du développement, mais nul chef de projet (ou Product Owner) n'aime lancer de chantier sans avoir une idée des délais. Il faut toujours avoir à l'esprit qu'ajouter des tâches de refactoring au Backlog (liste des chantiers en cours et à venir) impose forcément d'en mettre d'autres de côté.

• Quel est le coût estimé ?

Si cette question a parfois moins d'intérêt que celle des délais, elle peut devenir importante dans certains contextes : si la tâche implique la présence d'un sous-traitant expert pendant 10 jours, l'impact sur le budget sera différent d'une tâche de développement classique en interne.

• Existe-t-il des risques pour le produit ?

Les bugs sont courants dans les logiciels. Modifier le code d'une fonctionnalité qui a été testée et fonctionne correctement, c'est prendre le risque d'introduire un "comportement non désiré". Avez-vous bien maîtrisé l'impact de vos modifications ?

Un responsable de produit réduira ce risque en ajoutant des testeurs qui pourront détecter un les bugs. Mais le risque peut parfois être trop important lorsque l'on s'attaque à de vieilles applications : par exemple, aurez-vous toujours à disposition des testeurs maîtrisant le produit et pouvant approuver son bon fonctionnement ?

• Quel est le bénéfice apporté ?

Si l'on parle souvent de Retour sur Investissement (ROI), il est parfois difficile d'estimer le gain de temps et/ou d'argent qu'un refactoring pourrait apporter.

Par exemple : pour un logiciel rarement éteint et nécessitant 2 minutes pour se lancer sur un serveur, alléger le code pour le faire démarrer en 10 secondes semble être une perte de temps. Et pourtant, le développeur qui le relancera une vingtaine de fois par jour pour tester son code pourra gagner plus de 30 minutes par jour.

Si d'un autre côté, vous améliorez l'architecture pour permettre l'ajout de nouvelles fonctionnalités en un temps beaucoup moins important, le bénéfice ne sera pas présent si aucune nouvelle fonctionnalité n'est développée.

Répondre à cette question : c'est donc se demander : est-ce que le gain apporté (rapidité, souplesse, etc.) sera mis à profit plus tard ?

• Quels sont les impacts si rien n'est fait ?

Il est souvent nécessaire de mettre de côté certaines tâches de refactoring le temps de terminer des chantiers plus prioritaires. Mais rajouter des fonctionnalités et donc du code avant d'effectuer le refactoring aura-t-il un impact ? On peut se reposer toutes les questions précédentes en se demandant : si, plus on attend, plus l'impact changera *(il faudra évidemment faire varier la période selon le contexte)* :

- Quel sera le temps nécessaire si l'on attend 2 mois ?
- Quel sera le coût estimé si l'on attend 2 mois ?
- Quels seront les risques sur le produit si l'on attend 2 mois ?
- Quels seront les bénéfices si l'on attend 2 mois ?

Se poser ces questions permet souvent de se rendre compte qu'attendre n'est pas une bonne solution, mais peut parfois également remettre en question des tâches de refactoring.

Méthodologies

Nous avons maintenant toutes les clés en main pour quelques exemples. Une fois de plus, il est important de rappeler qu'il n'y a pas de bonne manière de faire mais il existe cependant des cas d'école et des outils pour vous aider !

Vous pouvez chercher manuellement dans vos projets du code dupliqué, des méthodes trop longues, ou encore des variables non utilisées… Mais pourquoi ne pas le faire automatiquement ?

Il est intéressant de commencer avec un "linter". C'est un outil à installer, ou disponible par défaut dans votre IDE. Après un peu de configuration, il vous permet de mettre en place des règles de qualité de code et de vous donner les conseils adéquats en temps réel. **Figure 1**

Lorsque vous commencez à réécrire une fonctionnalité, assurez-vous que cette dernière soit testée unitairement. Il serait bête d'apporter une régression lors du refactoring. Si ce n'est pas fait, commencez par-là, c'est le bon moment pour mettre en place du Test-Driven Development ! Écrire des tests unitaires va également vous permettre de comprendre le besoin auquel le code doit répondre.

Commencez par réduire la complexité du code, enlevez le superflu et ne gardez que l'essentiel pour passer les tests unitaires. Cela ne doit pas impacter le fonctionnel et vous devez simplement supprimer ce qui n'est pas ou plus utilisé !

Vos tests passent toujours ? Bien, vous pouvez maintenant renommer vos fonctions, méthodes, ou variables. Essayez de rester cohérent avec vos règles de nommage, les méthodes asynchrones terminent bien par « async » ? Une méthode qui retourne une valeur commence bien par "get" ?

EXEMPLE PRATIQUE

1 - JavaScript : lecture du code

Travaillons sur le code JavaScript suivant permettant de récupérer le total d'un panier :

```
const sampleBasket = [
  {
    name: 'banana',
    quantity: 5,
    price: 0.4,
    size: 0.5,
  },
  {
    name: 'apple',
    quantity: 2,
    price: 0.6,
    size: 0.7,
  },
];

const getTotal = (basket) => {
  let total = 0;
  for (let index = 0; index < basket.length; index += 1) {
    const item = basket[index];
    total += item.quantity * item.price
    + Math.max(item.quantity * item.size * 1.5, 10)
    + (item.quantity * item.price * 0.2);
  }
  return total;
};

console.log(getTotal(sampleBasket));
```

Voilà ce qui peut être obtenu après une séance de refactoring :

```
const getPrice = (item) => item.price * item.quantity;
const getShippingCost = (item) => Math.max(item.quantity * item.size * 1.5, 10);
const getTaxes = (item) => item.quantity * item.price * 0.2;

const getTotal2 = (basket) => basket
  .reduce((total, item) => total
    + getPrice(item)
    + getShippingCost(item)
    + getTaxes(item), 0);
```

L'accent a été mis sur la lecture du code et sa maintenabilité, le calcul a été découpé en plusieurs fonctions et la boucle a été remplacée par "Array". Le comportement est identique, mais il est dorénavant plus facile d'isoler le calcul des taxes ou des frais de port. Cela permet de mieux les tester, mais également de modifier plus facilement le comportement sans impacter d'autres fonctionnalités.

L'important est de ne pas tomber dans les pièges faciles du refactoring. Votre but n'est pas de réduire le nombre de lignes, mais de réduire la complexité !

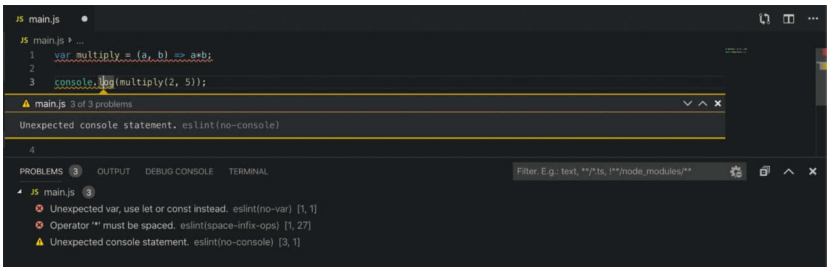


Figure 1

2 - Automatiser la qualité du code

Les exemples qui vont suivre sont des étapes de refactoring aboutissant à garantir une meilleure qualité du code de façon durable. Ces exemples sont simples, mais applicables à de nombreux projets complexes en impactant au minimum le comportement métier que peuvent avoir vos différentes classes.

Il est important de se rappeler que le refactoring est plus simple lorsqu'il est découpé en un ensemble d'étapes plus petites apportant des avantages sur le code et l'application.

2.1 - C# : utiliser les bons patterns

Dans de nombreux langages orientés objet, il est nécessaire de faire attention aux dépendances entre les différentes classes utilisées. Plus vous créerez de services, plus les dépendances entre ceux-ci pourraient devenir complexes.

L'inversion de contrôle est un pattern d'architecture qui vous permet de ne plus avoir à gérer le flot d'exécution de votre application. Une des conséquences est de ne plus avoir à se soucier des dépendances entre vos classes : celles-ci seront résolues par le framework.

En C#, .NET Core est capable de gérer pour vous l'injection des dépendances. Nous n'allons pas détailler ici le fonctionnement de ce pattern, mais vous trouverez à la fin de cet article un lien expliquant son utilisation. L'injection de dépendance et plus généralement l'inversion de contrôle sont implémentées dans de nombreux autres langages.

Prenons l'exemple ci-dessous : votre application console demande à votre service métier de récupérer un identifiant puis l'affiche.

```
static void Main(string[] args)
{
    DataProvider dataProvider = new DataProvider();
    MyService businessService = new MyService(dataProvider);

    Console.WriteLine($"Data ID : {businessService.GetNewDataIdentifier()}");
}
```

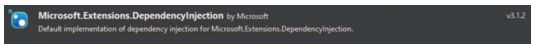
Votre classe métier MyService récupère une donnée de DataProvider et retourne son identifiant :

```
public class MyService
{
    private DataProvider _dataProvider;

    1 reference | 0 changes | 0 authors, 0 changes
    public MyService(DataProvider dataProvider)
    {
        this._dataProvider = dataProvider;
    }

    1 reference | 0 changes | 0 authors, 0 changes
    public Guid GetNewDataIdentifier()
    {
        return this._dataProvider.GetData().Identifier;
    }
}
```

Enfin, votre provider de données génère aléatoirement une nouvelle donnée à chaque appel de GetData() :

```
public class DataProvider
{
    1 reference | 0 changes | 0 authors, 0 changes
    public DataModel GetData()
    {
        return new DataModel
        {
            Identifier = Guid.NewGuid()
        };
    }
}
```

En .NET Core, il existe un package NuGet permettant de les gérer pour nous les dépendances entre services :



Une fois ajouté, nous pouvons modifier notre méthode Main() pour :

- Configurer l'injection de dépendance et la laisser instancier nos services ;
- Récupérer notre service prêt à l'emploi.

Voici ce qui est obtenu grâce à cette méthode :

```
static void Main(string[] args)
{
    var serviceCollection = new ServiceCollection();

     serviceCollection.AddTransient<DataProvider>();
     serviceCollection.AddTransient<MyService>();

    var serviceProvider = serviceCollection.BuildServiceProvider();
    var businessService = serviceProvider.GetService<MyService>();

    Console.WriteLine($"Data ID : {businessService.GetNewDataIdentifier()}");
}
```

Vous remarquez que l'objet « businessService » n'a jamais été instancié : notre objet serviceProvider l'a fait pour nous et lui a fourni l'ensemble des dépendances dont il avait besoin (à savoir le DataProvider)

2.2 - Modifier les tests unitaires

La classe MyService était testée par le code ci-dessous :

```
public class OldTestMyService
{
    [Test]
    0 references | 0 changes | 0 authors, 0 changes
    public void When_Get_ShouldReturnDataIdentifier()
    {
        var expectedValue = Guid.NewGuid();
        var dataProvider = new MockDataProvider(expectedValue);

        var businessService = new MyService(dataProvider);
        Assert.AreEqual(businessService.GetNewDataIdentifier(), expectedValue);
    }

    2 references | 0 changes | 0 authors, 0 changes
    public class MockDataProvider : DataProvider
    {
        private Guid _identifier;

        1 reference | 1/1 passing | 0 changes | 0 authors, 0 changes
        public MockDataProvider(Guid identifier)
        {
            this._identifier = identifier;
        }

        4 references | 0/1 passing | 0 changes | 0 authors, 0 changes
        public override DataModel GetData()
        {
            return new DataModel
            {
                Identifier = this._identifier
            };
        }
    }
}
```

Si vous êtes habitué au C#, vous noterez que ce test nécessitait que la méthode DataProvider.GetData soit virtuelle.

Avant l'injection de dépendance, nous devions d'abord savoir comment créer MyService, et lui créer une classe de provider de données spécifique pour le test.

Maintenant que nous n'avons plus à nous soucier des dépendances grâce à l'exemple précédent, nous pouvons grandement simplifier le simplifier. Mais avant cela, nous devons rendre notre classe MyService encore plus indépendante de la classe DataProvider en référençant plus directement la classe DataProvider, mais son interface mère IDataProvider :

```
public class MyService
{
    private IDataProvider _dataProvider;

    0 references | 0 changes | 0 authors, 0 changes
    public MyService(IDataProvider dataProvider)
    {
        this._dataProvider = dataProvider;
    }

    1 reference | 0 changes | 0 authors, 0 changes
    public Guid GetNewDataIdentifier()
    {
        return this._dataProvider.GetData().Identifier;
    }
}
```

Nous allons enfin pouvoir tester notre classe en lui donnant un objet « Mock » de IDataProvider, c'est-à-dire une fausse classe qui n'existera que pour notre test (nous utiliserons le framework NUnit ainsi que le package AutoMoqCore pour le faire). Voilà à quoi ressemblera notre test une fois nettoyé des dépendances à DataProvider :

```
[Test]
0 references | 0 changes | 0 authors, 0 changes
public void When_Get_ShouldReturnDataIdentifier()
{
    var mocker = new AutoMoqer();
    var expectedValue = new DataModel{ Identifier = Guid.NewGuid() };

    mocker.GetMock<IDataProvider>()
        .Setup(prov => prov.GetData())
        .Returns(expectedValue);


    var businessService = mocker.Create<MyService>();
    Assert.AreEqual(businessService.GetNewDataIdentifier(), expectedValue.Identifier);
}
```

Nous avons gagné en lisibilité. On peut ne tester qu'une méthode à la fois. Le refactoring précédent nous a donc permis d'améliorer nos tests et d'en implémenter de nouveaux plus précis et plus efficaces.

Bilan

Au fil de l'évolution des technologies et de vos connaissances, vous trouverez toujours des choses à modifier pour améliorer votre code, produit ou processus. Vous pourrez même utiliser des outils pour vous aider à trouver d'autres axes d'amélioration.

Le refactoring est indispensable pour accompagner la vie de vos projets, sans quoi de mauvaises pratiques peuvent s'accumuler et devenir un vecteur de risque trop important dans le maintien et la sécurité de l'application.

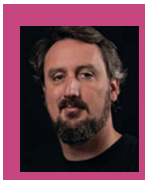
Gardez à l'esprit que le refactoring est contraignant en termes de coût, de risque et de temps. Pour ces raisons, il est important de se poser les bonnes questions avant de commencer.

Sources :

https://refactoring.com/catalog/
https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-3.1

K3S sur un cluster de Raspberry Pi

Afin de présenter à des étudiants les avantages et inconvénients d'une architecture microservices, j'ai voulu déployer sur un cluster de Raspberry Pi 3 B des instances de k3s, une version allégée de Kubernetes, pour illustrer les capacités de scalabilité et de résilience d'un orchestrateur de conteneurs. Je vous propose de m'accompagner dans l'installation et la configuration de celui-ci, avec son lot de déconvenues et, je l'espère, d'astuces qui pourront vous aider si vous tenter l'expérience vous-même.



Jérémy Lejeune
Concepteur Développeur à Zenika Nantes

zenika
<animés par la passion>

Installation de l'OS

Afin de profiter d'un OS maintenu à jour avec la Raspbian Lite (distribution Linux très légère à destination des processeurs ARM) et disposant d'outils de configuration et de conteneurisation, le choix s'est porté sur HypriotOS. Cette distribution orientée conteneurs comprend cloud-init qui m'a été fort utile pour l'initialisation et la configuration du Wi-Fi. L'utilisation de l'outil flash, fourni par Hypriot, avec l'ajout d'un hostname et l'utilisation d'un fichier user-data personnalisé a permis de démarrer les différents Raspberry et de les connecter à notre réseau local afin de poursuivre l'installation.

Exemple d'une commande pour flasher l'image sur la carte SD :

```
flash -n master -u wifi-user-data.yaml https://github.com/hypriot/image-builder-rpi/releases/download/v1.11.0/hypriotos-rpi-v1.11.0.img.zip
```

Le fichier wifi-user-data.yaml est une version modifiée de https://github.com/hypriot/flash/blob/master/sample/wifi-user-data.yml avec nos paramètres Wi-Fi. Préciser le hostname avec l'option -n. Il surcharge l'information contenue dans le fichier user-data.

Pour mon installation, j'ai utilisé les noms suivants pour différencier les nodes du cluster: master, node01, node02, node03 et node04. L'avantage de les nommer est de pouvoir les retrouver très simplement via ssh pour lancer l'installation de k3s, comme nous allons le voir.

Mise en place du serveur k3s

Mettons maintenant en place le serveur sur le node master. Il faut se connecter en ssh à celui-ci : ssh pirate@master.local (password: hypriot)

Les identifiants et mots de passe peuvent être changés via le fichier user-data à l'initialisation du Raspberry. J'ai laissé les valeurs par défaut pour des raisons de simplicité dans un premier temps, mais il est évidemment nécessaire de créer des utilisateurs avec des mots de passe différents et robustes pour chacun des nodes. Par défaut les Raspberry sont accessibles via ssh avec <identifiant>@<hostname>.local tel que nous les avons configurés précédemment.

Le script d'installation fourni par Rancher sur le repo Github de k3s permet de récupérer automatiquement l'exécutable adapté à l'architecture ainsi que divers utilitaires et scripts pour gérer le service :

```
curl -sfL https://get.k3s.io | sh -
```

LE MATÉRIEL

Le matériel que j'ai utilisé (un kit Pico 5), initialement prêté à @eric_briand pour expérimenter de l'éco-conception logicielle, se compose de :

- 5 Raspberry Pi 3B et des cartes SD 64Go pour l'OS et les systèmes de fichiers
- une alimentation externe
- un switch réseau pour connecter les nodes
- une prise RJ45 externe pour se connecter au switch en Ethernet

La prise réseau externe fut particulièrement utile à des fins de configuration ou pour se connecter directement au master sans passer par le réseau local, comme nous le verrons après.

Le tout est présenté dans un boîtier en plaques acryliques à monter soi-même, ce qui permet d'apprécier les lumières clignotantes du cluster.

Il est recommandé de jeter un œil sur le script ***install.sh*** *(https://github.com/k3s-io/k3s/blob/master/install.sh) avant de l'exécuter dans un shell sur votre machine.*

Le lancement du serveur k3s va être ajouté en tant que service dans systemd, ce qui permettra son démarrage automatique au branchement du Raspberry ou lors d'un reboot.

On notera l'existence d'un script *k3s-killall.sh*, qui arrête tous les processus système liés à k3s afin de relancer proprement l'exécutable, une commande que j'aurais beaucoup utilisée lors de l'installation et de la configuration des agents k3s.

Mise en place des nœuds k3s

Afin d'installer et de configurer correctement k3s sur les autres Pi, il nous faut récupérer le token généré par le serveur à son installation, ainsi que l'adresse IP de celui-ci. Et c'est à ce moment-là que les ennuis ont commencé pour moi… La récupération du token est aisée, une simple commande **sudo cat /var/lib/rancher/k3s/server/node-token** sur le master l'affiche. Nous le passerons en variable d'environnement au lancement de l'agent. Pour la récupération de l'IP du serveur c'est une autre histoire. Il m'a fallu de nombreux tâtonnements et moult **ifconfig** sur le master avant de comprendre que je devais utiliser l'adresse associée au Wi-Fi pour connecter les agents au master :

```
ifconfig wlan0 | grep -w inet | awk '{print $2}'
```

```
HypriotOS/armv7: pirate@master in ~
$ ifconfig wlan0 | grep -w inet | awk '{print $2}'
192.168.1.157
```

La raison est que k3s utilise un certificat généré à l'installation du master pour sécuriser la communication avec l'API et que celui-ci est exposé uniquement sur l'adresse wlan, empêchant la connexion depuis une autre adresse. La commande pour l'installation des agents est la suivante, à répéter sur chaque Raspberry restant :

```
curl -sfL https://get.k3s.io | K3S_URL=https://<ip_wlan_du_noeud>:6443 K3S_TOKEN=<node_token> sh -
```

Ces commandes devraient lancer le téléchargement et la configuration des agents. Une fois terminé il est possible de vérifier le bon fonctionnement de ceux-ci en se connectant sur le master et en lançant la commande :

```
sudo kubectl get nodes
```

```
HypriotOS/armv7: pirate@master in ~
$ sudo kubectl get nodes
NAME      STATUS   ROLES    AGE   VERSION
master    Ready    master   27d   v1.16.3-k3s.2
node03    Ready    <none>   25d   v1.16.3-k3s.2
node02    Ready    node     25d   v1.16.3-k3s.2
node01    Ready    node     25d   v1.16.3-k3s.2
node04    Ready    <none>   25d   v1.16.3-k3s.2
```

On notera que j'ai commencé à assigner des labels au différents nœuds et master à l'aide de la commande ***sudo kubectl label node <hostname> kubernetes.io/<master_or_node>*** *depuis le master.*
Il est possible (ce fut le cas pour moi à de multiples reprises…) que malgré une installation sans problème, ou lors du redémarrage du cluster, les nodes ne parviennent pas à se connecter au master, pour des raisons que j'ignore. Ma solution, un peu cavalière, j'en conviens, mais d'une efficacité redoutable : se connecter en ssh au node et ~~reboot!!~~ lancer les commandes suivantes :

```
sudo k3s-killall.sh
sudo k3s agent -server https://<ip_wlan_du_noeud>:6443 -token <node_token>
```

A ce sujet, il devient fastidieux de changer de machine en ssh ou de naviguer entre les différents terminaux dans tmux, aussi je vous propose d'installer kubectl et de le configurer pour accéder à notre cluster. Pour pouvoir agir sur le master de façon transparente, il nous reste à récupérer la configuration du cluster sur celui-ci, modifier l'adresse du serveur et nommer le fichier **.kube/config** dans votre répertoire personnel. J'ai utilisé **scp** pour copier le fichier via ssh :

```
scp pirate@master.local:/etc/rancher/k3s/k3s.yaml .
```

Figure 1

Vous devrez remplacer '127.0.0.1' par l'adresse ip wlan du master à ligne « **server : https://127.0.0.1:6443** »

Figure 2

Puis renommer et déplacer le fichier :

```
mv k3s.yaml ~/.kube/config
```

Cela va nous permettre de lancer toutes les commandes depuis notre machine favorite, sans **sudo** et nous simplifiera la vie pour lancer et accéder au Dashboard.

Figure 1

```
~/Documents/Développement/Zenika/cluster-raspberry ▷ scp pirate@master.local:/etc/rancher/k3s/k3s.yaml .
pirate@master.local's password:
k3s.yaml                                      100% 1052    97.0KB/s   00:00
```

Figure 2

```
T3htZVA5dWRGVmVhSDV1WGJJeWRXNlYyMHNNNjdyN3hNVm5QNytRdWw5ZWpJekFoTUE0R0ExVWREd0VCL3dRRQpBd0lDcERBUEJnTlZIUk1CQWY4RUjJQOG43dUFJaEFJU0lqRzJmeS9jbHd3aitsTGIyNTNBRlZiNDMKYVRib0hURUpCL1dJUDNqeAotLS0tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==
    server: https://192.168.1.157:6443
  name: default
contexts:
- context:
    cluster: default
    user: default
  name: default
current-context: default
kind: Config
preferences: {}
```

Figure 3

```
~/Documents/Développement/Zenika/cluster-raspberry ▷ kubectl get all -n kubernetes-dashboard
NAME                                        READY   STATUS    RESTARTS   AGE
pod/kubernetes-dashboard-5996555fd8-sc9cw   1/1     Running   5          12d
pod/dashboard-metrics-scraper-76585494d8-qnttf   1/1     Running   18         15d

NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
service/dashboard-metrics-scraper   ClusterIP   10.43.56.243    <none>        8000/TCP   15d
service/kubernetes-dashboard        ClusterIP   10.43.135.58    <none>        443/TCP    15d

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/dashboard-metrics-scraper   1/1     1            1           15d
deployment.apps/kubernetes-dashboard        1/1     1            1           12d

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/dashboard-metrics-scraper-76585494d8   1         1         1       15d
replicaset.apps/kubernetes-dashboard-5996555fd8        1         1         1       12d
```

Installation du Dashboard

Commençons par déployer les éléments nécessaires avec la commande :

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard
/v2.0.0-beta8/aio/deploy/recommended.yaml
```

Et vérifions que tout s'est déployé correctement :

```
kubectl get all -n kubernetes-dashboard
```

Figure 3

Nous devons ensuite créer un ServiceAccount et un Role pour avoir le droit d'accéder au Dashboard et d'effectuer des modifications ou des déploiements sur le cluster :

```
cat >> dashboard-adminuser.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
 name: admin-user
 namespace: kubernetes-dashboard
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: admin-user
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: cluster-admin
subjects:
- kind: ServiceAccount
 name: admin-user
 namespace: kubernetes-dashboard
EOF
```

Puis appliquer les modifications :

```
kubectl apply -f dashboard-adminuser.yaml
```

Il nous faut ensuite utiliser la commande **kubectl proxy &**, ce qui va exposer celui-ci à l'adresse suivante, accessible dans notre navigateur web : http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/

Attention, le dashboard ne sera accessible que depuis la machine sur laquelle la commande de proxy a été lancée.

Reste à se connecter à celui-ci avec le token lié au serviceAccount créé précédemment. La commande suivante permet de récupérer le token :

```
kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes
-dashboard get secret | grep admin-user | awk '{print $1}')
```

Figure 4

Copiez et collez la valeur de celui-ci dans le champ "*Saisissez un jeton*" : **Figure 5**

You're in, well done! On peut maintenant aller vérifier la bonne santé de nos nodes. **Figure 6**

Il reste à trouver un magnifique projet microservice à déployer sur notre cluster tout neuf ! Pour l'instant je n'ai pas trouvé de projet de démo simple à déployer, les projets existants comme Sock shop ou Hipster Shop de GoogleCloudPlatform utilisent des fonctionnalités de Kubernetes non présentes ici… Je ne désespère pas de réussir à modifier subtilement les manifests pour faire fonctionner l'un des deux.

Pour conclure

A posteriori, je me rends compte qu'une meilleure connaissance des problématiques d'adressage réseau et du fonctionnement interne du cluster m'aurait probablement évité de m'arracher les cheveux sur la connexion des agents au serveur.

Après consultation des collègues, une solution à envisager pour éviter de dépendre du réseau local et d'une connexion Wifi est de se connecter en Ethernet et d'utiliser son poste de travail comme serveur DHCP. Cela permettrait de s'abstraire de la configuration du Wi-Fi faite en début d'article et de s'assurer de toujours pouvoir communiquer avec le cluster.

Enfin malgré tous mes efforts, je n'ai toujours pas réussi à adapter un des projets cités ci-dessus à k3s, mes pods tombant presque tous dans l'enfer du *CrashLoopBackoff*. Malgré ces écueils, l'installation est assez aisée et permet de monter rapidement un environnement de démonstration pour les fonctionnalités d'un orchestrateur et les avantages et inconvénients d'une architecture microservices.

Un grand merci à @pyaillet et @eric_briand pour leurs conseils avisés.

Sources:

https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard

https://github.com/kubernetes/dashboard/blob/master/docs/user/access-control/creating-sample-user.md

Figure 4

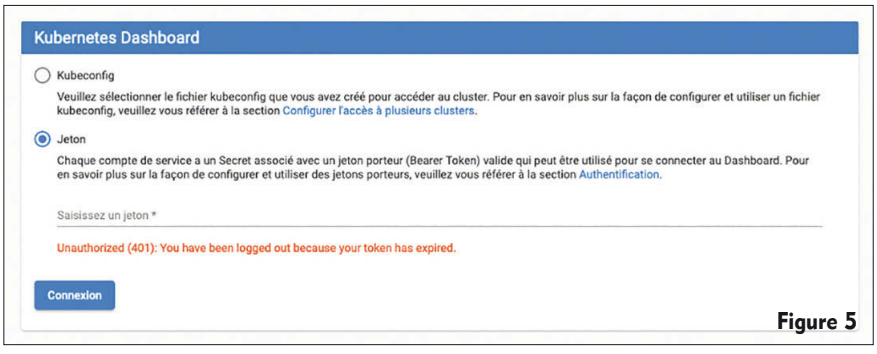


Figure 5

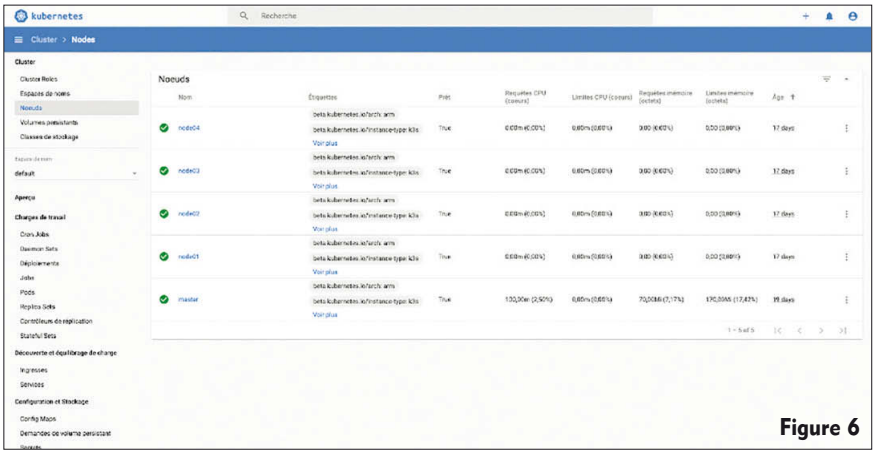


Figure 6

Du code legacy vers CQRS : est-ce possible ?

Dorra Bartaguiz

Dorra est développeuse Azure/.NET depuis plus de 12 ans. Coach technique, elle fait aussi partie de l'équipe de formateurs d'Arolla. Elle partage son savoir-faire en publiant des articles sur notre blog, animant des conférences et des meetups. Dorra a aussi enseigné dans une école d'ingénieur à Paris pendant plusieurs années.

Pendant son expérience, elle a acquis des valeurs qui s'articulent autour du clean code, le craft et l'agilité qu'elle adore partager.

J'imagine que vous avez déjà entendu parler du pattern CQRS, de ses bienfaits, de son implémentation … Si vous ne le connaissez pas, ce n'est pas grave vous êtes au bon endroit pour le découvrir. D'expérience, nous travaillons majoritairement sur des applications qui ont vécu, donc je vous propose de partir d'un code existant et le factoriser pour arriver à l'implémentation de ce pattern en douceur.

Les exemples sont en C#, mais ne vous inquiétez pas, ils sont faciles à lire, enfin je l'espère.

CQRS quésaco ?

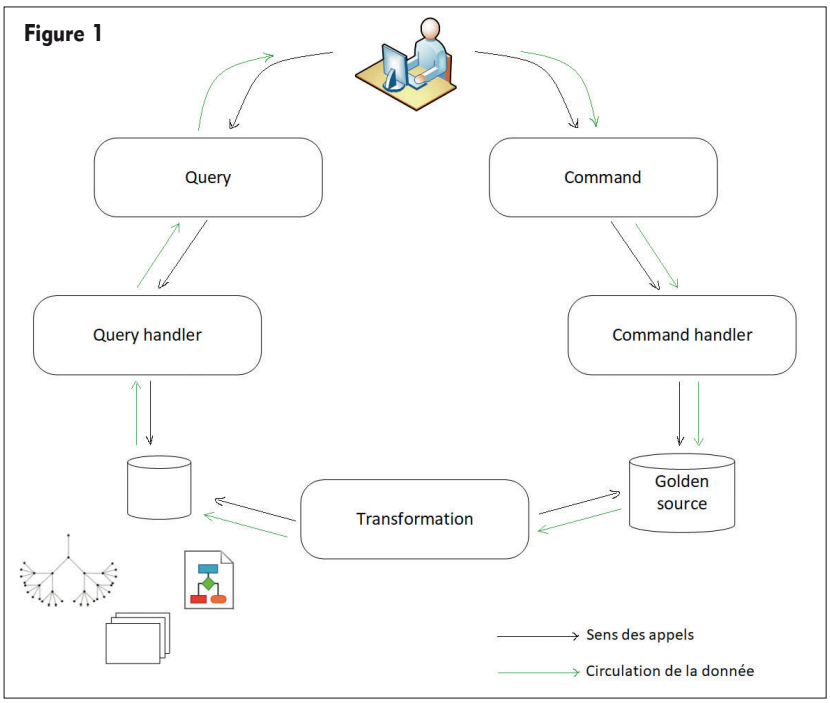
CQRS (Command and Query Responsibility Segregation) est un pattern de séparation des opérations de lecture et d'écriture des informations. Il est présenté pour la première fois par Greg Young.

Traditionnellement, les modèles utilisés à la fois pour la lecture et l'écriture ne correspondent pas forcément à tous les cas d'utilisation, notamment ceux de lectures récurrentes de grappes d'objets ou de lectures massives de données. Comme les besoins ne sont pas forcément les mêmes entre l'écriture (write model) et la lecture (read model) des données, CQRS propose alors la séparation des deux modèles et donc de modéliser différemment les données de l'écriture et les données de la lecture. On peut donc utiliser une base relationnelle par exemple pour l'écriture et un graphe comme modèle de lecture. Mais il faut toujours garder en tête que la base d'écriture est notre base de confiance ou de référence (appelée aussi « Golden source »).

Deux termes reviennent souvent quand on parle de CQRS :

- Une commande (command) correspond à une action d'écriture ou de modification des données.
- Une requête (query) porte la demande de lecture. Elle retourne la donnée correspondante à ce qui est attendu sans superflu et elle ne doit jamais modifier les données.

Figure 1



Techniquement, les solutions de mise en place de ce pattern sont multiples. On peut imaginer l'utilisation d'une même base de données, mais en dématérialisant le schéma relationnel en créant des vues, comme on peut imaginer une séparation physique des bases de données, voire l'utilisation, comme on l'a évoqué plus haut, de différents types de base de données.

La « golden source » est la base de référence. Elle correspond à la base d'écriture et sera utilisée comme base de confiance au cas où une défaillance de synchronisation arrive lors de la transformation des données.

La démarche

Souvent, quand on présente ce pattern, on le présente sur un kata en mode « green field ». Ce mode consiste à développer le kata de zéro (« from scratch »), sans aucune base de code existante. Pour cet article, j'ai décidé de le présenter sous une autre forme, en mode « brown field ». Je vais donc partir d'une base de code existant que je vais refactorer pour arriver à l'implémentation du pattern. Vous trouverez l'implémentation sur mon GitHub https://github.com/iAmDorra/BankKata_and_cqrs.

Ma base de code est sécurisée avec des tests unitaires et des tests d'acceptance. Ces tests représentent mon harnais de sécurité en cas d'erreur de refactoring. Pour en savoir plus sur les tests je vous invite à lire mon article sur le sujet : https://www.arolla.fr/blog/2020/01/differents_tests_pour_developpeur/

Pour simplifier la lecture des commits, j'ai créé une branche par étape d'implémentation comme décrit dans le schéma ci-dessous. Le tout est mergé dans master au fur et à mesure de l'avancement du développement. **Figure 2**

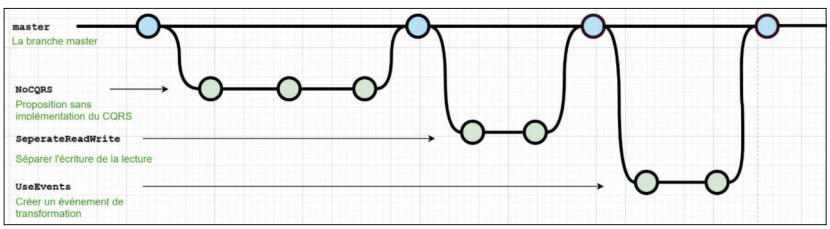


Figure 2

Je veux aussi signaler que j'ai mis le focus sur l'implémentation du pattern CQRS, donc j'ai omis exprès beaucoup de recommandations du DDD (Domain Driven Design).

Bank Kata

Le principe du kata est simple. L'idée est de pouvoir suivre les transactions d'un compte bancaire. On peut donc déposer de l'argent (deposit), retirer de l'argent (withdraw) et voir l'état du compte à tout moment (balance statements). Pour simplifier l'utilisation de ce service, il vaut mieux que les transactions soient dans un ordre chronologique décroissant. La dernière transaction effectuée doit être donc affichée en premier avec l'état de la balance globale. Et comme un exemple est plus clair pour expliquer les choses, en voici un sous le format « gherkin » :

Given a client makes a *deposit* of *100* on *01/01/2020*
And a *deposit* of *200* on *02/01/2020*
And a *withdraw* of *50* on *03/01/2020*
When he prints his account statement
Then he would see:

Date	Amount	Balance
03/01/2020	-50	250
02/01/2020	200	300
01/01/2020	100	100

Proposition sans CQRS

La branche NoCQRS correspond comme son nom l'indique à une proposition d'implémentation sans CQRS. Dans cette proposition, j'ai créé un service (BankingService) qui permet de faire un dépôt (Deposit) et un retrait (Withdraw) et retourne la balance (PrintBalance). Il s'agit de trois méthodes différentes. Les deux méthodes Deposit et Withdraw permettent de rajouter des transactions de dépôt et de retrait. Ces transactions sont gardées en mémoire au niveau de « Transactions » qui ne sont pas dans le domaine, mais dans une couche que j'ai nommé « Application » puisque c'est une implémentation technique. **Figure 3**

À chaque demande de l'état du compte, on récupère tout l'historique des transactions au format enregistré et on les parcourt pour recréer de nouveaux objets (statements) qui correspondent à notre besoin. Pourquoi ne sépare-t-on pas les deux concepts ?

Refactoring pour implémenter CQRS

Séparation de l'écriture et de la lecture

Le pattern CQRS nous invite à séparer la partie lecture de la partie écriture étant donné que les deux modèles sont différents. J'ai donc créé une nouvelle branche (SeperateReadWrite) partant du dernier commit de la branche NoCQRS.

L'idée est donc d'avoir la classe « Transactions » qui permet de rajouter les transactions de dépôt et de retrait et une autre classe « BalanceRetriever » qui retourne l'état de la balance au moment de l'appel.

La classe « Transactions » contiendra aussi notre golden source. Pour simplifier, je garde tout en mémoire, mais on peut imaginer une base SQL derrière par exemple. Il faut juste se dire que ce choix est un problème technique non prioritaire pour l'instant.

La classe « BalanceRetiever », quant à elle, retourne le relevé du compte. Par contre avant de la créer directement, je vais d'abord préparer le terrain pour minimiser la taille du refactoring à faire.

Dans un premier temps, j'extrais la méthode GetStatements dans la classe « BankingService » ensuite je la bouge dans « Transactions » pour enfin l'extraire dans la classe « BalanceRetriever ». **Figure 4**

Une fois ces étapes faites, je me retrouve avec une dépendance de « BalanceRetriever » à « Transactions ». À ce niveau, réellement, rien n'a changé dans le traitement. On recalcule le relevé à partir de tout l'historique. Et si j'utilise les événements pour envoyer l'information en temps réel au « BalanceRetriever » lors d'un ajout d'une nouvelle transaction. Comment pourrai-je faire ?

Utilisation des événements

J'ai créé une branche à part pour cette modification que j'ai nommé « UseEvents ». Ce n'est pas vraiment original comme nom de branche.

L'idée est d'ajouter un événement dans « Transactions » et le lancer à chaque ajout. Et ensuite, il n'y a plus qu'à refactorer « BalanceRetriever » pour capturer cet événement et ajouter les lignes du relevé au fur à mesure des ajouts. L'utilisation des événements permet d'enlever la dépendance et de réduire le couplage. **Figure 5**

Pour information, j'ai laissé la classe « TransactionEvent Args », qui encapsule la transaction ajoutée, au niveau du projet

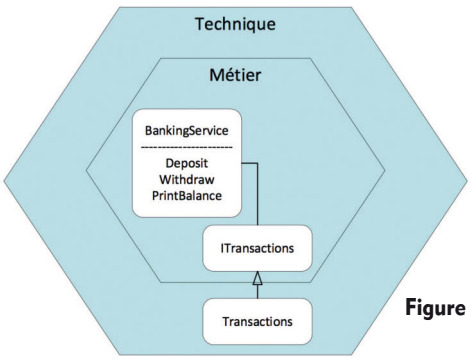


Figure 3

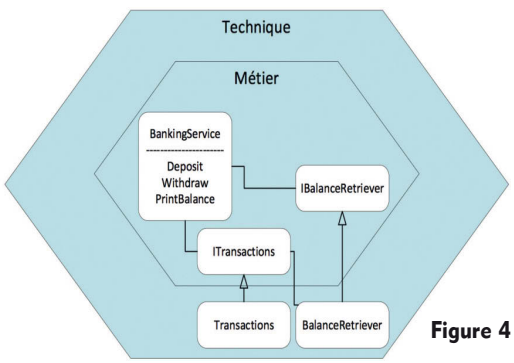


Figure 4

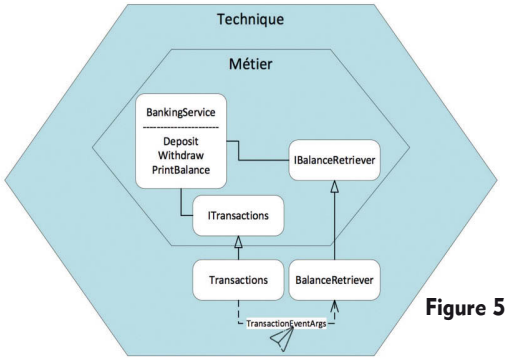


Figure 5

« Banking.Application », car je considère que c'est technique. D'ailleurs les contrats de mon domaine n'ont pas changé.

Problème réglé

À ce niveau, on peut très bien se satisfaire de la solution. Le **schéma 6** représente la solution actuelle. D'un côté, à droite du schéma, on a la partie d'écriture (write model) avec la transaction à ajouter et le handler de l'ajout et à gauche la partie lecture (read model) avec le calcul de la balance qui se base sur l'événement d'ajout de transaction et le relevé du compte (List<AccountStatement>).

Piège à éviter

Le schéma correspond bien à ce qu'on veut implémenter dans le CQRS, à savoir la séparation de la partie lecture de celle de l'écriture. Par contre, on peut facilement tomber dans certains pièges.

D'ailleurs si on regarde le code, on va se rendre compte qu'une des règles métier est de faire la somme des transactions pour avoir la balance. Mais quand on cherche où est implémentée cette règle, on se rend compte qu'on était tellement focus sur le refactoring pour obtenir une implémentation du pattern qu'on a déporté accidentellement les règles métiers en dehors du domaine. Avant de continuer l'amélioration de l'implémentation du pattern, il faut remonter la règle de gestion dans le domaine.

Il faut faire attention à l'étape de transformation, car ça peut correspondre à une règle métier et donc doit être placée dans le domaine, et c'est notre cas. Étant donné que le calcul correspond à additionner le montant de la transaction et la valeur de la balance actuelle, le traitement peut, très bien, être placé dans la classe « Transaction » puisqu'on va accéder à ses données internes. Pour le suivi des modifications, j'ai créé une branche correspondante à ce refactoring nommée « MoveCalculationToDomain ».

Certes la modification est anodine, mais il faut faire attention, car c'est très important de déterminer et de garder les règles métier dans le domaine pour séparer les contextes et les responsabilités de chaque élément.

Lors des lectures, le modèle d'écriture n'est pas sollicité (réduit la contention), on évite des agrégations au moment de la lecture qui peuvent être coûteuses si la quantité de données est importante. On gagne en indépendance des modèles.

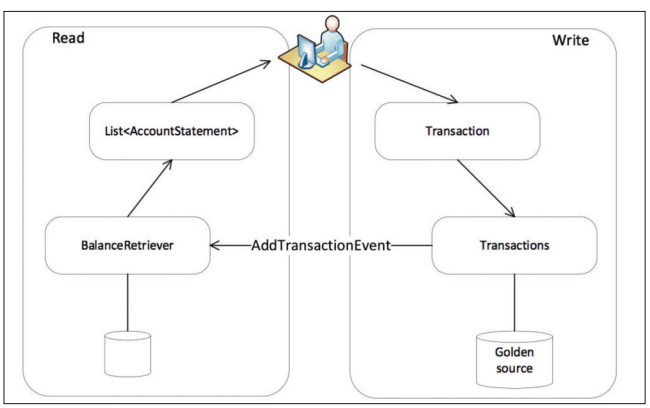
Plus loin encore

Une fois le pattern implémenté, on se rend compte de la réduction de la contention lors de la lecture. En effet, on présente un modèle simple en évitant les agrégations qui peuvent être coûteuses si les données sont volumineuses. Ainsi, on gagne en indépendance des modèles et chacun évolue à son rythme.

Pour aller plus loin dans l'implémentation du pattern, on peut très bien imaginer un « service bus » dans lequel on envoie les messages correspondants aux transactions ajoutées. Donc le repository « Transactions » va créer le message et l'envoyer dans ce bus et « BalanceRetriever », qui est à l'écoute (le « listener ») de ce bus, va récupérer les messages au fur et à mesure pour recalculer la balance.

Dans certains cas, on peut considérer l'écriture et la lecture comme deux domaines différents et donc mettre une ACL (« Anti-Corruption Layer ») entre les deux. Jérémie Chassaing en parle d'ailleurs dans un interview :
https://www.youtube.com/watch?v=wu5Z8yXpBCM

Schéma 6



Avant de finir

Je suis partie d'un exemple simple pour le factoriser et arriver à l'implémentation du pattern. Mon but est de montrer que le pattern est plutôt simple à implémenter. Mais même avec un exemple simple, on peut tomber dans des pièges. Et ces pièges sont multiples :

- Certain·e·s développeu·r·se·s trouvent ce pattern tellement intéressent qu'elles/ils confondent pattern et architecture et veulent l'implémenter pour l'ensemble de l'application. Gardons en tête que c'est un pattern donc on peut l'implémenter pour une fonctionnalité de l'application.
- Pour une application type CRUD (« Create, Read, Update, Delete ») par exemple, l'implémentation de CQRS n'apporte que de la complexité inutile à l'application, car on n'a pas forcément besoin d'optimiser le temps de la recherche qui est généralement unitaire (recherche d'une entité ou quelques lignes de données).
- Enfin, parfois vouloir implémenter le CQRS dans une architecture déjà complexe ou vouloir le combiner avec d'autres patterns comme l'« event-sourcing » peut rendre le pattern difficile à assimiler pour les membres de l'équipe et donc les évolutions peuvent devenir fastidieuses au fil du temps. Je vous conseille de ne l'implémenter que quand vous avez des performances médiocres au niveau de la lecture massives des données, autrement, vous risquez de rajouter de la complexité inutile qui peut engendrer des dettes techniques accidentelles par la suite.
- Il vaut mieux partir sur une solution alternative et ensuite la factoriser pour arriver à l'implémentation du CQRS que l'implémenter au démarrage du projet/fonctionnalité et se rendre compte que le code devient un château de cartes ; dès qu'on touche à une brique, on engendre plusieurs régressions.

Pour information

Si vous vous demandez comment j'ai réalisé le graphe des branches, j'ai utilisé le template de Bryan Braun :
https://gist.github.com/bryanbraun/8c93e154a93a08794291df1fcdce6918.
Très pratique

Conclusion

J'espère que ça devient plus clair pour vous. Si ce n'est pas le cas ou si vous n'êtes pas d'accord avec mes choix, je compte sur vous pour me contacter en mettant un commentaire ou via les réseaux sociaux afin d'échanger et de partager nos points de vue. À bientôt.

Programmation & calculatrices

PARTIE 2

Philippe nous a codé un superbe dossier sur la programmation sur les calculatrices. Depuis plusieurs années, elles supportent le langage Python et certaines peuvent même servir à de la robotique, à interagir avec capteurs, etc. C'est un support idéal pour apprendre les bases de la programmation. La rédaction.

Dossier codé et compilé par
Philippe BOULANGER
Manager des expertises C/C++ et Python
www.invivoo.com

INVIVOO BEYOND TECH

Polynômes

Les polynômes sont des fonctions riches en fonctionnalités. Elles sont utiles dans de nombreux domaines. Ils font partie ainsi des fonctions les plus étudiées dans le cursus du lycée et au-delà.

$A X^2 + B X + C = 0$

Explications

Sans se lancer dans de longues démonstrations, la recherche des racines de cette équation dépend du signe du discriminant :

$\Delta = b^2 - 4\ a\ c$

- Si Δ est nul, alors il y a une racine unique : $x = -b / (2a)$
- Si $\Delta > 0$ alors l'équation admet 2 solutions réelles :

$x_1 = (-b + \sqrt{\Delta}) / (2a)$

$x_2 = (-b - \sqrt{\Delta}) / (2a)$

- Si Δ est strictement négatif alors l'équation admet 2 racines complexes :

$x_1 = (-b + \sqrt{\Delta}) / (2a)$ avec qui $\sqrt{\Delta}$ est un nombre complexe

$x_2 = (-b - \sqrt{\Delta}) / (2a)$ avec qui $\sqrt{\Delta}$ est un nombre complexe

En Python

Voici la version qui ne retourne que les racines réelles : on retourne None s'il n'y a pas de solutions réelles sinon on retourne un tuple contenant 2 racines.

```
from math import sqrt

def racine_polynome2( a, b, c ):
  d = b * b - 4 * a * c
  if d < 0:
    return None
  else:
    return ( ( -b + sqrt( d ) ) / 2 / a,
         ( -b - sqrt( d ) ) / 2 / a )
```

Pour le cas où l'on souhaite des racines dans l'espace des nombres complexes, le code Python s'en trouve simplifié :

```
from cmath import sqrt as csqrt

def racine_polynome2_complexe( a, b, c ):
  d = b * b - 4 * a * c
  return ( ( -b + csqrt( d ) ) / 2 / a,
       ( -b - csqrt( d ) ) / 2 / a)
```

Représentation d'un polynôme

Le type Polynôme n'existe pas dans Python, pour travailler avec les polynômes. Il nous faudra donc trouver une structure de stockage adaptée. Un polynôme est défini par la liste de ses coefficients où chacun d'entre eux est associé à une puissance de x.

Une liste Python, qui est aussi un tableau, contient des éléments qui sont indexés de 0 à *n-1* (où est *n* est le nombre d'éléments). On pourrait stocker le coefficient a_i correspondant à x^i dans l'élément de la liste Python indicé *i*. « $x^5-x^4+2x^3-2x^2+3x-3$ » sera ainsi codé : [-3, 3, -2, 2, -1, 1].

Indices de la liste	0	1	2	3	4	5
Monômes	x^0	x^1	x^2	x^3	x^4	x^5
Coefficients du polynôme	**-3**	**3**	**-2**	**2**	**-1**	**1**

Cette forme de représentation sera adoptée pour tous les algorithmes que nous décrirons dans la suite de cet article.

Évaluation de polynôme : la méthode de Hoerner

Explications

Toutes les opérations sur un ordinateur ont un coût que l'on compte en nombre de cycles d'horloge. Les multiplications sont plus coûteuses que les additions et le calcul de la puissance d'un nombre est beaucoup plus coûteux qu'une multiplication. Calculer x^{10} est donc consommateur en puissance de calcul ; heureusement un mathématicien du nom de Hoerner a trouvé une méthode permettant de diminuer le coût de l'évaluation en minimisant le nombre d'opérations.

Voici un exemple sur un polynôme de degré 4 que nous noterons $P_4(x) = a\,x^4 + b\,x^3 + c\,x^2 + d\,x + e$. Par un jeu de factorisation des opérations, on peut le réécrire en $P_4(x) = (((a\,x + b)\,x + c)\,x + d)\,x + e$. Par ce biais, on remplace les calculs de puissances par des additions et multiplication successives.

Algorithme

L'algorithme du schéma de Hoerner est assez simple à mettre en œuvre. On utilisera la variable P pour le polynôme. La fonction « longueur » retourne le nombre d'éléments du tableau :

```
Fonction EvaluerPolynome( P, x )
        longueur( P ) - 1 -> n
        0. -> v
        Pour i de n à 0 avec pas de -1
                v * x + P[ i ] -> v
        Fin Pour
        Retourne v
Fin Fonction
```

Code Python

Python fournit un grand nombre de moyens de parcourir un tableau notamment de la fin vers le début grâce à la fonction ***reversed*** qui fournit un itérateur (un moyen de parcourir les éléments du tableau du dernier élément vers le premier).

```
def EvaluerPolynome( P, x ):
  e = 0.
  for c in reversed( P ):
    e = e * x + c
  return e
```

Calculer la dérivée d'un polynôme

Explications

La dérivée du monôme ax^n est nax^{n-1}. Et la dérivée de $f(x)+g(x)$ est $f'(x)+g'(x)$. Dès lors, automatiser le calcul de la dérivée d'un polynôme complexe est une tâche simple.

Algorithme

```
Fonction Derivation( P )
        [] -> D
        longueur( P ) - 1 -> n
        Pour i de 1 à n
                ajouter (i * P[i]) à la fin de D
        Fin Pour
        Retourne D
End Fonction
```

Programme Python

Nous allons profiter de la concision des listes par compréhension alliée à la fonction ***enumerate*** sur un sous-tableau de P pour en faciliter l'écriture.

```
def DerivationPolynome( P ):
  return [ i * c for i, c in enumerate( P[ 1: ], 1 ) ]
```

Intégration numérique

L'intégrale de f(x) sur l'intervalle [a, b] est la somme algébrique (aire signée) des aires délimitées par le graphe de f et de l'axe (Ox). Les parties au-dessus de l'axe (Ox) sont comptées positivement, celles en-dessous seront comptées négativement. De nombreuses méthodes numériques existent pour fournir une valeur approchée d'une intégrale qui ont toutes en commun de subdiviser [a,b] en n intervalles de largeur égale et d'approximer f(x) par une fonction (souvent un polynôme) donc le calcul de l'aire est plus simple.

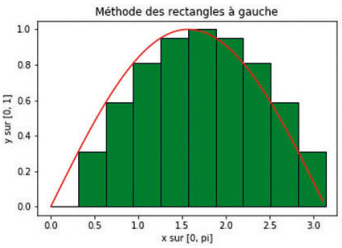
La méthode des rectangles

Explications

Dans cette méthode on approchera ***f(x)*** sur l'intervalle ***[a,b]*** par un segment de droite parallèle à l'axe (Ox) créant ainsi un rectangle : dont le calcul de l'aire est simple. Le choix du segment est un élément important qui influera sur l'algorithme et sa rapidité de convergence.

Rectangles à gauche

Dans cette méthode on approchera ***f(x)*** sur l'intervalle $[a_n, b_n]$ par le segment défini par l'équation : $y=f(a_n)$



Dans le cas présent, si $h=(b-a)/n$, nous aurons une valeur approchée de l'intégrale I_n :

$$I_n = h\sum_{i=0}^{n-1} f(a+ih)$$

Pour nous faciliter l'écriture, nous allons combiner des fonctions facilitatrices :

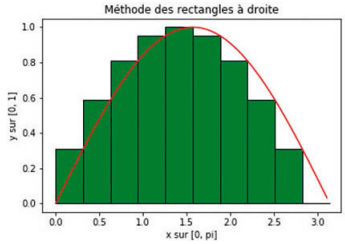
- sum(vec)
  calcule la somme des éléments de l'itérable ***vec***. Sur une version PC nous lui préférerions la fonction *math.fsum*. Par itérable on entend une séquence que l'on peut parcourir comme une liste.
- map(f, vec)
  retourne f(x) pour chaque x de vec

Voici le code Python permettant de calculer l'aire de ces rectangles :

```
from math import *
def rectangle_a_gauche( f, a, b, n ):
    h = ( b - a ) / n
    s = sum( map( f, ( a + h * i for i in range( n ) ) ) )
    return s * h
print( rectangle_a_gauche( sin, 0., tau, 100 ) )
```

Rectangles à droite

Dans cette méthode on approchera ***f(x)*** sur l'intervalle $[a_n, b_n]$ par le segment défini par l'équation : $y=f(b_n)$



Dans le cas présent, si $h=(b-a)/n$, nous aurons une valeur approchée de l'intégrale I_n :

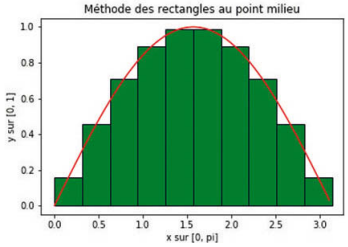
$$I_n = h\sum_{i=1}^{n} f(a+ih)$$

Pour nous faciliter l'écriture, nous allons combiner les mêmes fonctions facilitatrices que pour la méthode des rectangles à gauche. Pour générer les entiers de 1 à ***n*** inclus, nous utiliserons l'appel suivant : ***range(1, n+1)***. Voici le code Python permettant de calculer l'aire de ces rectangles :

```
from math import *
def rectangle_a_droite( f, a, b, n ):
    h = ( b - a ) / n
    s = sum( map( f, ( a + h * i for i in range( 1, n+1 ) ) ) )
    return s * h
print( rectangle_a_droite( sin, 0., tau, 100 ) )
```

Rectangles aux points milieux

Dans cette méthode on approchera ***f(x)*** sur l'intervalle $[a_n, b_n]$ par le segment défini par l'équation : $y=f((a_n+b_n)/2)$.



Dans le cas présent, si $h=(b-a)/n$, nous aurons une valeur approchée de l'intégrale I_n :

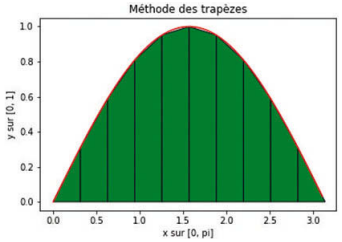
$$I_n = h\sum_{i=0}^{n-1} f(a+ih+\frac{h}{2})$$

Voici le code Python permettant de calculer l'aire de ces rectangles :

```
from math import *
def rectangle_point_milieu( f, a, b, n ):
    h = ( b - a ) / n
    h2 = 0.5 * h
    s = sum( map( f, ( a + h2 + h * i for i in range( n ) ) ) )
    return s * h
print( rectangle_ point_milieu( sin, 0., tau, 100 ) )
```

La méthode des trapèzes

Dans cette méthode on approchera ***f(x)*** sur l'intervalle ***[a,b]*** par un segment de droite reliant le point de coordonnées (a, f(a)) au point de coordonnées (b, f(b)) créant ainsi un trapèze dont le calcul de l'aire est simple : ***(f(a) + f(b)) * (b – a) / 2***.



On peut donc approcher l'intégrale I_n de f sur [a, b] par :

$$I_n = h\left(\frac{f(a)+f(b)}{2}+\sum_{i=1}^{n-1} f(a+ih)\right)$$

Ce qui se traduit en Python par :

```
def trapeze( f, a, b, n ):
    h = ( b - a ) / n
    s = 0.5 * ( f( a ) + f( b ) ) + \
      sum( map( f, ( a + h * i for i in range( 1, n ) ) ) )
    return s * h
```

La méthode de simpson

On interpole ***f*** sur l'intervalle ***[a, b]*** par un polynôme de degré 2 passant par les 2 points extrémités et par le point milieu ***(m, f(m))***. Ce polynôme est construit à l'aide des polynômes de Lagrange (voir https://fr.wikipedia.org/wiki/Interpolation_lagrangienne). L'équation de ce polynôme est :

$$P(x) = f(a)\frac{(x-m)(x-b)}{(a-m)(a-b)} + f(m)\frac{(x-a)(x-b)}{(m-a)(m-b)} + f(b)\frac{(x-a)(x-m)}{(b-a)(b-m)}$$

On en déduit que l'intégrale de f est :

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left[f(a)+4f(\frac{a+b}{2})+f(b)\right]$$

En découpant l'intervalle ***[a, b]*** en ***n*** sous-intervalles de la même taille $h=(b-a)/n$:

$$\int_a^b f(x)dx \approx \frac{h}{6}\sum_{i=0}^{n-1}\left(f(a+ih)+4f(a+ih+\frac{h}{2})+f(a+(i+1)h)\right)$$

En calculant cette suite I_n, on se rend compte que certains termes sont répétés :

$$I_n = \frac{h}{6}\left[f(a)+2\sum_{i=1}^{n-1} f(a+ih)+4\sum_{i=1}^{n-1} f(a+ih+\frac{h}{2})+f(b)\right]$$

Nous pouvons simplifier cette formule en :

$$I_n = \frac{h}{6}\left[f(a)+2\sum_{i=1}^{2n-1}\left((1+i \bmod 2)* f(a+i\frac{h}{2})\right)+f(b)\right]$$

Que nous pouvons traduire en Python par :

```
def simpson2( f, a, b, n ):
    n *= 2
    h = ( b - a ) / n
    s = f( a ) + f( b )
    s += 2 * sum( ( 1 + ( i % 2 ) ) * f( a + i * h ) for i in range( 1, n ) )
    return s * h / 3
```

Zéros de fonctions

Rechercher les valeurs de x pour lesquelles f(x)=0 est un cas courant de problèmes mathématiques que les étudiants ont souvent à résoudre.

LA DICHOTOMIE

Explications

Le point de départ de cette méthode est le théorème de Bolzano encore appelé « théorème des valeurs intermédiaires ». Il indique que si une fonction continue sur un intervalle qui prend deux valeurs *m* et *n*, alors elle prend toutes les valeurs intermédiaires entre *m* et *n*.

Une conséquence directe du théorème de Bolzano est :

Soit ***f*** une fonction de $\mathbb{R}$ dans $\mathbb{R}$, elle est continue et strictement monotone sur l'intervalle [***a***, ***b***].

$f(a)*f(b) \leq 0 \Rightarrow (\exists\, r \in [a, b], f(r)=0)$

La dichotomie ou bissection (*bisection* en anglais) est une méthode stable qui garantit la convergence vers une valeur approchée de ***r*** dès lors que les hypothèses sur ***f*** sur [***a***, ***b***] sont bien respectées. La méthode va consister à réduire l'intervalle de recherche en le divisant par 2 à chaque itération. Par récurrence, l'erreur absolue sera au maximum de $2^{-n-1}*(b-a)$. Le défaut de cette méthode est d'avoir une convergence lente par rapport à la méthode de Newton ou celle de la sécante.

Algorithme

On a un intervalle $[a_n, b_n]$ avec $f(a_n)*f(b_n)<0$.

On calcule le milieu de l'intervalle $m=(a_n + b_n)/2$.

Si $f(a_n)*f(m)<0$ alors $r \in [a_n, m]$ sinon $r \in [m, b_n]$.

Programme Python

On ne va pas stocker une suite de valeurs, on va se contenter de mettre à jour la valeur de a ou de b successivement. *epsilon* sera la valeur d'arrêt de l'algorithme.

```
from math import cos

def bisection( f, a, b, epsilon=1.E-6 ):
    while b - a > epsilon:
        m = 0.5 * ( a + b )
        if f( a ) * f( m ) <= 0:
            b = m
        else:
            a = m
    return 0.5 * ( a + b )
r = bisection( cos, 0.5, 2, 1.0E-10 )
print( "f(", r, ") =", cos(r) )
```

L'algorithme converge en 34 itérations dans le cas de l'exemple sous la fonction.

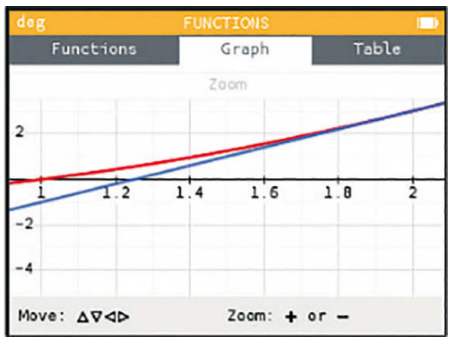
La méthode de newton-raphson

Explications

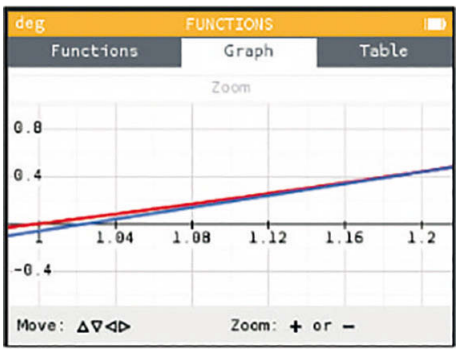
La méthode de Newton est basée sur l'approximation locale de la fonction ***f*** par son développement de Taylor-Young à l'ordre 1. Supposons que nous sommes autour d'un point x_0 proche de la racine recherchée. En utilisant l'approximation de ***f*** (qui se trouve être une droite), on va pouvoir trouver l'intersection de cette droite avec l'axe (Ox) et obtenir un point x_1 qui permet de se rapprocher de la racine.

Dans l'exemple suivant, nous allons travailler sur la fonction $f(x)=x^2-1$. Les racines de ce polynôme sont évidentes : $r_1=-1$ et $r_2=1$. Cependant si on lui applique l'approximation de Taylor-Young en $x_0=2$, nous obtenons sur la calculatrice Numworks :

- La courbe ***f(x)*** en rouge
- La tangente à ***f*** au point ***x=2*** est en bleu



L'équation de la tangente est $T_0(x)=4x-5$. Et le zéro de la tangente se produit pour $x_1=5/4$. Si on recommence l'algorithme en x_1, on trouve $T_1(x)=5x/2-41/16$. Et donc le zéro de la tangente se produit en $x_2=41/40$.



Cette méthode a le mérite de converger rapidement si l'on part d'un point proche de la solution.

L'algorithme

Le développement de Taylor-Young en x_0 de ***f*** nous donne :

$$f(x) = f'(x_0)(x - x_0) + f(x_0)$$

Donc la valeur de x_1 se calculera grâce à la formule suivante :

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Par définition la valeur de la dérivée de ***f*** en ***x*** se calcule comme suit :

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}$$

Une bonne approximation de la dérivée pourra donc être obtenue en utilisant une valeur de ***h*** suffisamment petite (10^{-6} par exemple). On peut ainsi construire une suite de nombres x_n à partir d'une valeur de x_0 de la manière suivante :

$$x_{n+1} = x_n - 2h \frac{f(x_n)}{f(x_n + h) - f(x_n - h)}$$

On s'arrêtera si $|f(x_n)| < \varepsilon$ avec ε suffisamment petit car on aura une bonne approximation de la racine recherchée. Mais pour éviter le risque de divergence, nous allons adjoindre un compteur d'itérations qui, s'il dépasse une valeur définie à l'avance, on arrête la boucle et on sort en erreur.

```
Fonction Newton( f, x, h=10^-6, epsilon=10^-8 )
    99 -> NbIterationMax
    0 -> n
    Tant que ( |f(x)| > epsilon ) et ( n < NbIterationMax )
        x - 2 * h * f( x ) / (f(x+h) - f(x-h)) -> x
        n+1 -> n
    Fin Tant que
    Si n=NbIterationMax Alors
            Sors en erreur
        Sinon
            Retourne x
    Fin Si
Fin Fonction
```

En recherchant Newton(cos, 0.5) l'algorithme converge en 4 itérations, ce qui est très efficace en comparaison de la dichotomie.

Code Python

Au lieu de sortir en erreur, en Python, nous retournerons *None*.

```
from math import fabs

def Newton( f, x, h=1.0E-6, epsilon=1.0E-8 ):
    NbIterationMax = 99
    n = 0
    while ( fabs( f(x) ) > epsilon ) and ( n < NbIterationMax ):
        x = x - 2 * h * f( x ) / ( f( x + h ) - f( x - h ) )
        n += 1
    return x if n < NbIterationMax else None
```

La méthode de la sécante

Explications

L'idée de la méthode de la sécante est très simple : pour une fonction f continue et monotone sur un intervalle ***[a, b]***, on trace le segment ***[AB]*** où ***A=(a, f(a))*** et ***B=(b, f(b))***. On calcule ***M=(m, 0)*** le point d'intersection de la droite ***(AB)*** avec l'axe des abscisses. La droite ***(AB)*** s'appelle la sécante. Si ***f(a)*f(m)<0*** alors le 0 de ***f*** se trouvera sur ***[a, m]***, sinon il sera sur ***[m, b]***. Il suffit de réitérer sur le sous-intervalle jusqu'à ce que $|f(m)| < \varepsilon$ ou que $b-a < \varepsilon$.

L'algorithme

La valeur de m se calcule facilement :

$$m = a - f(a) \frac{b-a}{f(b) - f(a)}$$

Ce qui nous permet de construire l'algorithme suivant :

```
Fonction methode_secante( f, a, b, ε )
    Tant que b – a < ε faire
        a – f( a ) * ( b – a ) / ( f( b ) – f( a ) ) -> m
        f( m ) -> fm
        Si |fm| < ε alors
                Retourne m
            Sinon si f( a ) * fm < 0 alors
                m -> b
            Sinon
                m -> a
        Fin si
    Fin tant que
    Retourne ( a + b ) / 2
Fin fonction
```

Code Python

Transformons cet algorithme en programme Python :

```
def methode_secante( f, a, b, epsilon=1.0E-6 ):
    while b - a > epsilon:
        m = a - f( a ) * ( b - a ) / ( f( b ) - f( a ) )
        fm = f( m )
        if fabs( fm ) < epsilon:
            return m
        elif f( a ) * fm < 0:
            b = m
        else:
            a = m
    return 0.5 * ( a + b )
```

En le testant sur « methode_secante(cos, 0, 3, 1.0E-10) », on converge en 5 itérations et le résultat retourné est π/2 avec ses 16 décimales exactes.

JEUX

Les premiers programmes que j'ai écrits lorsque j'avais 13 ans étaient des programmes de jeux. Et tous les élèves veulent avoir des jeux (téléphone, tablette ou console portable). Petits exemples.

Trouver un nombre entre 1 et 100

Principes

La calculatrice choisit un nombre entier aléatoire entre 1 et 100. Le joueur (l'humain) a le droit à 10 essais. A chaque essai, le joueur propose un nombre et on lui indique si le nombre cherché a été trouvé ou s'il est plus grand ou plus petit.

Nombres aléatoires

Pour générer un nombre aléatoire entier, il existe un module ***random*** qui est disponible sur les différentes calculatrices du marché. La fonction qui nous intéresse est randint(a, b) où a et b sont 2 entiers tels que a<b. Cette fonction nous retournera un nombre entier ***n*** aléatoirement choisi entre a et b tel que a ≤ n ≤ b.

Programme

Commençons par les initialisations :

```
# initialisation
import random
nb_essai = 10
trouve  = False
numero  = random.randint( 1, 100 )
```


Maintenant, attaquons-nous à la boucle principale du jeu. La première chose à prendre en compte est la condition d'arrêt : on s'arrête dès que le joueur a trouvé la bonne solution ou dès que le nombre d'essais est expiré. Il ne faut donc pas oublier de décrémenter le compteur d'essai(s).

```
# boucle principale
while not trouve and nb_essai != 0:
  valeur = int( input( "Entrez un numero? " ) )
  if valeur == numero:
    print( "bravo vous avez trouve!" )
    trouve = True
  elif valeur < numero:
    print( " c'est plus grand" )
  else:
    print( " c'est plus petit" )
  nb_essai = nb_essai - 1
```

Et pour finir, n'oublions pas de fournir la solution si jamais le joueur n'a pas trouvé la bonne solution dans le nombre d'essais prédéterminés.

```
# on donne la solution si nécessaire
if not trouve:
  print( "La bonne valeur etait", numero )
```

Calcul mental

Principes

L'idée est de permettre de s'entraîner au calcul mental grâce à la calculatrice. Nous allons nous entraîner avec les tables de multiplications mais le même principe pourra être appliqué à des additions ou d'autres calculs. A chaque partie, le joueur aura 10 calculs à effectuer. Pour chaque calcul, nous allons choisir deux nombres de 2 à 10 et laisser le joueur nous donner le résultat de la multiplication. En cas d'erreur nous lui fournissons le bon résultat sinon nous le félicitons.

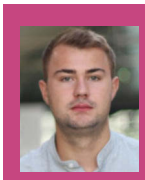
Programme

Le programme de jeu est :

```
import random

for i in range( 10 ):
  nb1    = random.randint( 2, 10 )
  nb2    = random.randint( 2, 10 )
  resultat = nb1 * nb2
  texte   = "{} x {} = ".format( nb1, nb2 )

  valeur = int( input( texte ) )
  if valeur == resultat:
    print( " bravo!" )
  else:
    print( " La bonne solution etait", resultat )
```

Bon code sur votre calculatrice.

Deep Racer : découvrir et optimiser l'apprentissage des modèles



Matthieu Rousseau

Date-Engineer co-fondateur de Modeo et Pronoo. Gagnant français de l'édition 2019 de la AWS DeepRacer League.

Lors du Re:Invent 2018, AWS a annoncé la sortie d'un nouveau service : DeepRacer. Ce service permet aux développeurs de tout niveau de découvrir l'apprentissage par renforcement de manière pratique. L'objectif est simple : entraîner un modèle d'apprentissage par renforcement pour participer à une course de voitures autonomes mondiale. J'ai eu l'occasion de participer à l'édition 2019 et de remporter l'édition française, me qualifiant ainsi pour la finale mondiale aux États-Unis en décembre 2019.

La voiture

AWS a conçu en partenariat avec Intel et OpenVino une voiture à l'échelle 1/18e. Elle dispose de deux caméras de 4 mégapixels positionnées à l'avant, un capteur Lidar sur son toit qui est capable de balayer à 360 degrés sur 12 mètres ainsi qu'un accéléromètre et un gyroscope. Son moteur lui permet d'aller jusqu'à 5m/s avec un angle de braquage maximal de 30 degrés.

Les caméras et le Lidar se connectent via un port USB. Il est donc possible de customiser la voiture. D'ailleurs, la première version de la DeepRacer ne possédait qu'une seule caméra. Le Lidar et la deuxième caméra ont fait leur apparition lors du Re:Invent 2019.

Elle se pilote via une application web embarquée sur la voiture, qui donne accès aux caméras en temps réel. C'est sur cette plateforme qu'on sélectionne son modèle, qu'on calibre la voiture et qu'on régule sa vitesse.

Le champ d'action de la voiture est limité et doit être renseigné par le développeur. Plus le nombre d'actions possibles est élevé, plus le modèle est complexe à entraîner. En revanche, un modèle dont le nombre d'actions est faible aura du mal à être rapide sur la piste et aura plus tendance à en sortir. **Figure 1**

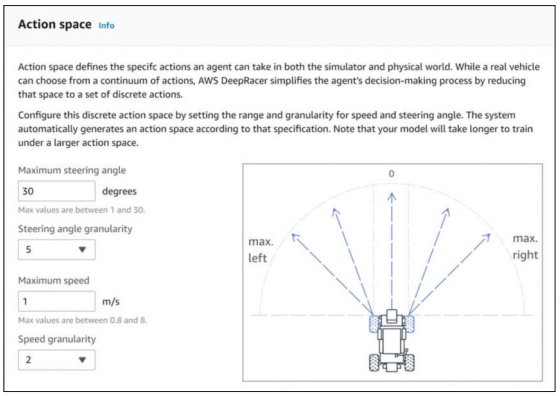


Figure 1

Les types de courses

Pour la saison 2018-2019, il n'existait qu'un seul type de course : le contre la montre (TT) où l'objectif est de réaliser le meilleur temps pour un tour de piste en moins de 3 minutes. Depuis 2020, de nouvelles courses ont été ajoutées. On peut maintenant faire concourir plusieurs voitures en même temps en face à face (H2H). Il existe également un mode course d'évitement d'objets (OA) disponible uniquement en virtuel. À partir de 2020, le programme par défaut du circuit virtuel comprend les épreuves TT, OA et H2H.

La compétition

La première édition de la DeepRacer League a eu lieu en 2018-2019. Elle a réuni plusieurs milliers de participants dans le monde, qui se sont affrontés avec un objectif commun : soulever la coupe à Las Vegas au Re:Invent. Pour se qualifier pour la finale, il y a deux possibilités : gagner une course physique ou une course virtuelle.

Les courses physiques ont lieu lors des AWS Summits. Ce sont des journées de conférences, de formations, de rencontres et de démos qui ont lieu une fois par an dans les plus grandes villes du monde. À cette occasion, AWS installe une piste et des DeepRacer pour permettre aux participants de s'affronter. Cette piste est identique à celle qu'on peut trouver dans la console d'entraînement DeepRacer. Les participants se présentent tour à tour avec leur meilleur modèle. Ils ont ensuite trois minutes pour réaliser le tour le plus rapide sur la piste. À l'issue de la journée, le développeur ayant effectué le meilleur temps remporte le Summit et gagne une invitation pour la finale mondiale à Las Vegas (tous frais payés par AWS). Ceux qui ont terminé dans le top 10 remportent une DeepRacer.

Pour les courses virtuelles, tout se passe dans la console AWS. Tous les mois, un nouveau circuit est disponible. Si un véhicule effectue trois tours consécutifs sur ce nouveau circuit dans l'environnement virtuel, la moyenne des 3 tours est intégrée au classement.

La façon de construire son modèle est très différente selon le type de compétition auquel on participe. Pour la course physique, on cherche à avoir un modèle capable de fonctionner sur n'importe quelle piste en prenant en compte les sources d'incertitudes (ombres sur la piste, voiture mal calibrée, etc.). Pour la course virtuelle, on va plutôt chercher à faire apprendre au modèle la piste, en guidant la voiture dans ses décisions pendant la phase d'entraînement. Ces modèles sont dédiés à une piste précise, mais ils permettent de faire des temps bien meilleurs que les modèles physiques. En revanche, il est très difficile pour eux de bien performer sur des pistes qu'ils n'ont jamais vues.

Au total, ce sont 64 développeurs venant des quatre coins du monde qui se sont qualifiés pour la finale à Las Vegas. **Figure 2**

L'architecture du service DeepRacer

DeepRacer est une combinaison de différents services :

1. Amazon Sagemaker : la plateforme d'AWS pour entraîner, tester et déployer des modèles de Machine Learning. C'est dans Sagemaker que s'opère l'entraînement et le test du modèle d'apprentissage par renforcement.
2. AWS RoboMaker : un service pour tester et déployer des applications de robotique dans des environnements virtuels en 3D totalement personnalisables. RoboMaker est utilisé pour créer le circuit dans lequel la voiture évolue.
3. Amazon S3 : le service de stockage objet d'Amazon. On l'utilise pour stocker les modèles entraînés et leurs artefacts (checkpoints, logs, etc.).
4. Amazon Kinesis : le service de traitement de flux vidéo en temps réel. Il est utilisé pour donner accès au développeur à une vidéo en temps réel du comportement de la voiture.
5. Amazon CloudWatch : le service de monitoring d'AWS. On y trouve donc tous les logs relatifs à l'exécution des différentes tâches. **Figure 3**

Le lexique de la DeepRacer et de l'apprentissage par renforcement

Reinforcement learning (apprentissage par renforcement) est une méthode de Machine Learning axée sur la prise de décision autonome d'un agent afin d'atteindre des objectifs spécifiques par le biais d'interactions avec un environnement. C'est la succession de tentatives et d'erreurs qui permet au modèle d'apprendre.

L'apprentissage du modèle repose sur une fonction de récompense qui gratifie chaque action prise par l'agent. Plus la récompense est grande, mieux l'agent s'est comporté.

Figure 2

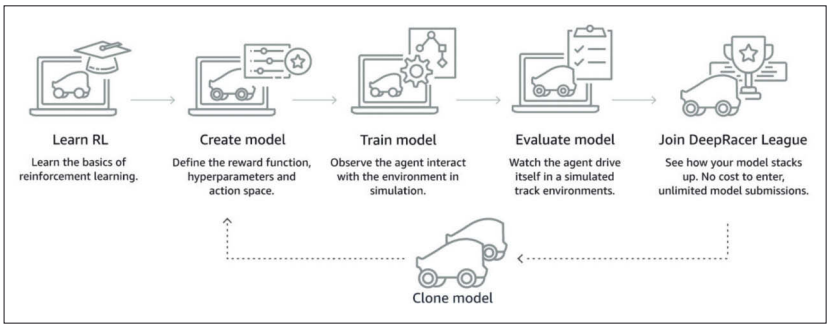
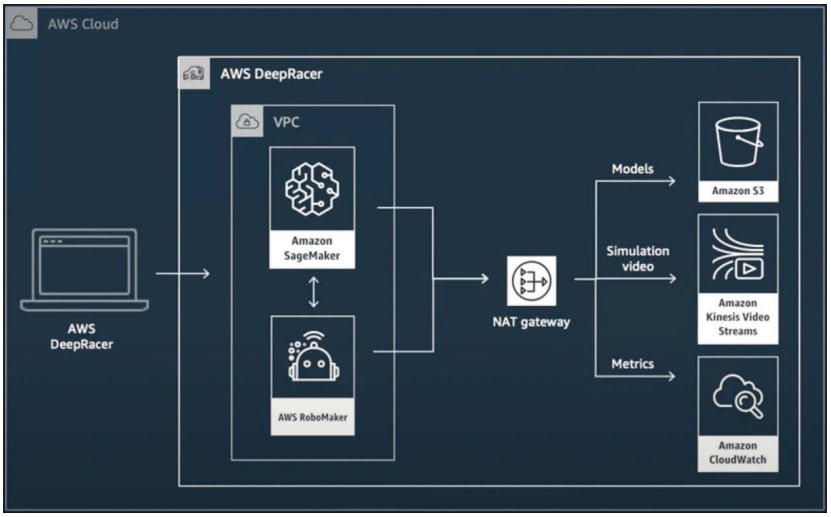


Figure 3



L'objectif de l'algorithme est de trouver les bons paramètres pour maximiser la fonction de récompense. Cette fonction est donc un élément crucial dans l'apprentissage par renforcement, car elle va définir le comportement de l'agent et sa capacité à apprendre rapidement.

Pour la DeepRacer, l'agent est la voiture, l'environnement correspond à la piste, les différents chemins possibles et les conditions de circulation. L'objectif est que le véhicule atteigne sa destination le plus rapidement possible, sans quitter la piste. Les récompenses sont des points utilisés pour encourager un déplacement sûr et rapide vers la ligne d'arrivée.

Modèle de Reinforcement learning

L'environnement dans lequel un agent évolue est conditionné par trois paramètres :

- Les différents états dans lesquels l'agent peut se situer ;
- Les actions que l'agent peut entreprendre ;
- Les récompenses qu'il reçoit en agissant.

La stratégie avec laquelle l'agent décide de son action est appelée « policy ». La policy prend en entrée l'état de l'environnement et prédit l'action à entreprendre (exemple : une roue de la voiture sort de la piste, la policy prédit qu'il faut tourner pour corriger la trajectoire).

Lorsqu'on fait rouler la voiture et qu'elle prend des décisions, c'est sur ce modèle que nous allons nous baser pour prédire la bonne action en fonction de l'état de l'agent.

Policy network

Un policy network est un réseau neuronal entraîné. Il prend des images vidéo en entrée et prédit la prochaine action de l'agent.

Optimization algorithm

L'objectif d'un modèle de Machine Learning est de correctement prédire Y, en fonction de X (par exemple X = une image de la route, Y = une décision de la voiture).

Pour être en capacité de prédire Y, un modèle de Machine Learning doit passer par une première phase d'entraînement. On confronte le modèle à des données pour lui faire prendre une décision. Pour chaque décision, on détermine à quel point il s'est trompé. Le rôle de l'algorithme d'optimisation est de modifier les paramètres (poids) pour minimiser le niveau d'erreur.

Pour l'apprentissage par renforcement, l'algorithme est optimisé en maximisant les récompenses futures grâce à la fonction de récompense.

Il existe de nombreux algorithmes d'optimisation, les plus connus sont : SGD, RMSProp, Adam…

Réseau de neurones (Neural network)

Un réseau de neurones est un système informatique inspiré du fonctionnement du cerveau. Il possède plusieurs nœuds (similaires à des neurones) reliés entre eux. Chaque nœud reçoit une entrée (stimulus), s'active si le signal d'entrée est suffisamment fort (activation) et produit une sortie en fonction de l'entrée et de l'activation. Ils sont très utilisés, car ils peuvent approximer n'importe quelle fonction.

Il existe de nombreux types de réseaux de neurones, les plus populaires sont les réseaux de neurones convolutifs (CNN), récurrents (RNN), les autoencodeurs et les Generative

Adversarial Network (GAN). Quand on parle aujourd'hui des dernières innovations dans l'intelligence artificielle comme la vision par ordinateur, le traitement de la voix ou du langage naturel, il s'agit généralement de ce type de système. **Figure 4**

Fonction de récompense (Reward function)

C'est un algorithme qui prend en compte l'état d'un agent dans un environnement et qui :

- Renvoie une bonne note lorsque l'action prise par l'agent est bonne et doit être renforcée ;
- Renvoie une mauvaise notes lorsque l'action prise par l'agent est mauvaise et doit être décourageé.

La fonction de récompense est un élément clé de l'apprentissage par renforcement. Elle détermine le comportement que l'agent apprend en encourageant des actions spécifiques par rapport à d'autres.

Épisode

C'est la période pendant laquelle l'agent recueille des expériences comme données d'entraînement dans son environnement. Pour la DeepRacer, le début d'un épisode correspond à un point de départ aléatoire sur la piste. L'épisode se termine quand la voiture quitte la piste ou atteint la ligne d'arrivée. Les différents épisodes peuvent avoir des durées différentes.

Itération

C'est l'ensemble des expériences consécutives entre chaque évolution de la policy. Le nombre d'épisodes par itération est un hyperparamètre qui doit être renseigné par le développeur. Après chaque itération, l'ensemble des expériences pour chaque épisode est stocké dans une liste. Le réseau de neurones est alors mis à jour en utilisant des échantillons aléatoires provenant de cette liste.

Job d'entraînement (Training job)

Le processus qui crée le modèle d'apprentissage par renforcement et ses artefacts. Pour chaque entraînement, on distingue deux sous-tâches :

- L'exploration de l'agent dans l'environnement virtuel, c'est-à-dire la succession d'épisodes et la collecte des données d'entraînement ;
- Le calcul du gradient basé sur les données d'entraînements collectées puis la mise à jour des poids du réseau de neurones.

Job d'évaluation (Evaluation job)

C'est le processus qui permet de tester les performances d'un modèle. Pour la DeepRacer, on teste le modèle en lui faisant faire des tours sur une piste dans l'environnement virtuel. La qualité du modèle se base sur deux données : le pourcentage de la piste réalisée et le temps mis pour faire un tour.

Hyperparamètre (Hyperparameters)

Un hyperparamètre est un paramètre non entraînable dont la valeur est fixée en amont de l'entraînement. Il va conditionner la façon dont le modèle apprend. Il s'agit par exemple du taux d'apprentissage (learning rate), de la taille des batchs de données ou encore le niveau d'entropie.

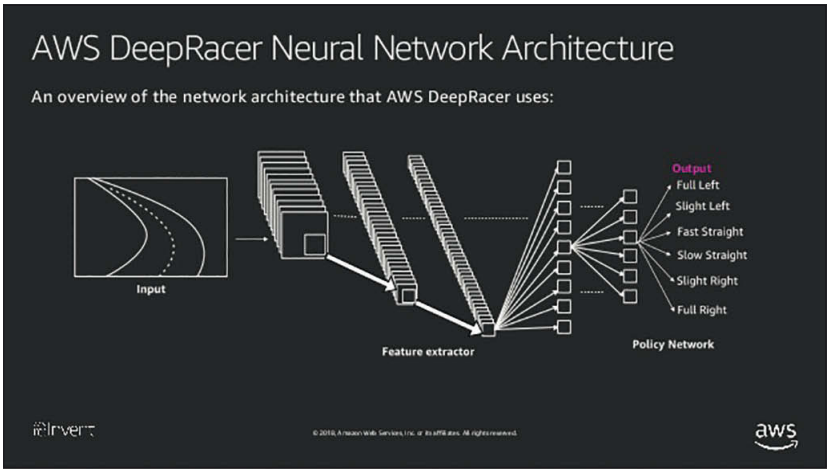


Figure 4

Gradient descent batch size

Pour chaque épisode on stocke chaque expérience dans une liste. Le gradient descent batch size est un hyperparamètre qui correspond au nombre d'expériences à sélectionner aléatoirement dans cette liste pour mettre à jour les poids du réseau de neurones. Un grand batch size permet de rendre les mises à jour des poids plus fluides et plus stables, mais peut rendre l'entraînement plus lent.

Number of epochs

Le nombre d'epochs est un hyperparamètre qui correspond au nombre de cycles pendant lesquels l'ensemble des données d'entraînements seront utilisées par le réseau de neurones. Par exemple pour une nombre d'epoch égal à 3, le réseau de neurones verra 3 fois les données d'entraînement. Plus le nombre d'epoch est grand, plus les mises à jour des poids seront stables, mais l'entraînement sera plus lent. Pour un petit batch size, il est préférable d'utiliser un petit nombre d'epochs.

Learning rate

Le learning rate est un hyperparamètre qui indique à quel point il faut modifier le modèle en réponse à l'erreur estimée chaque fois que les poids du modèle sont mis à jour. Le choix du learning rate est crucial, car une valeur trop faible peut rendre l'entraînement trop long, tandis qu'une valeur trop élevée peut rendre le modèle instable et l'empêcher de converger.

Entropy

C'est un coefficient d'incertitude qui détermine quand est-ce qu'il faut ajouter des décisions aléatoires à l'ensemble des expériences. Cette incertitude supplémentaire aide la DeepRacer à explorer plus largement l'espace d'action. Une valeur d'entropie plus importante encourage le véhicule à explorer l'espace d'action plus en profondeur.

Discount factor

Le discount détermine dans quelle proportion l'agent se soucie des récompenses dans un avenir lointain par rapport à celles étant immédiates. Plus la valeur du facteur d'actualisation est élevée, plus les expériences que le véhicule considère comme étant lointaines sont importantes et plus l'entraînement sera lent.

Loss type

C'est le type de fonction utilisée pour mesurer l'erreur de prédiction. C'est un élément essentiel pour mettre à jour les poids du réseau de neurones. Deux fonctions sont disponibles pour entraîner la DeepRacer : le Huber Loss et la Mean Squared Error Loss (MSE).

Un bon algorithme d'entraînement doit apporter des changements progressifs à la stratégie de l'agent pour qu'il passe progressivement de la prise de décisions aléatoires à la prise de décisions stratégiques augmentant ainsi sa récompense.

S'il apporte un changement trop important, l'entraînement devient instable et l'agent finit par être en incapacité d'apprendre. À mesure que les mises à jour des poids deviennent plus importantes, Huber prend des incréments plus petits par rapport à la MSE. Lorsque le modèle a du mal à converger, il vaut mieux utiliser Huber. Sinon la MSE rend l'entraînement plus rapide.

Waypoints

Dans l'environnement virtuel, la piste est échantillonnée par des waypoints. Ce sont des points de repère qui permettent de connaître le niveau de progression de la voiture sur la piste.

Number of experience episodes between each policy-updating iteration

Le nombre d'expériences requises avant de mettre à jour les poids. Pour des problèmes simples de renforcement et d'apprentissage, un batch avec peu d'expérience peut être suffisant et l'apprentissage sera relativement rapide. Pour des problèmes plus complexes, un batch d'expérience plus important sera nécessaire pour fournir des données non corrélées. Dans ce cas, l'apprentissage est plus lent, mais plus stable.

Création d'un véhicule personnalisé

Pour créer son véhicule personnalisé, il faut se rendre dans l'onglet My Garage du service DeepRacer dans la console AWS (https://console.aws.amazon.com/deepracer/home?region=us-east-1#garage).

Pour faire du Time Trial, une seule caméra suffit et le LIDAR n'est pas nécessaire.

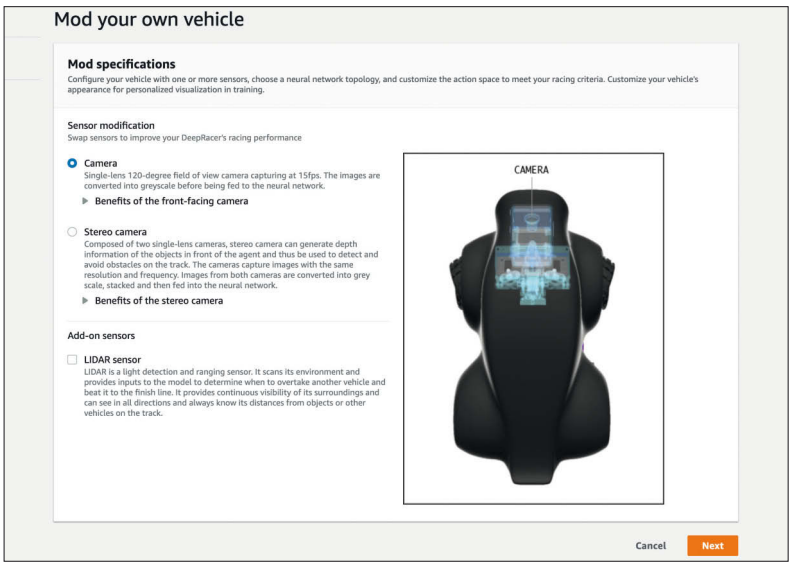
Une fois ces options sélectionnées, il reste à renseigner l'Action Space. Pour le Steering Angle, je le limite à 20 degrés pour éviter que la voiture prenne des virages trop brusquement.

Pour la granularité du Steering Angle, je la laisse à 5. Au vu de ce que j'ai pu tester, c'est un bon compromis entre un entraînement rapide et la possibilité pour la voiture de suivre des trajectoires complexes.

Pour la vitesse maximale, je la mets à 2,5 m/s. Si la voiture va trop lentement, il devient alors plus difficile d'évaluer ses prises de décisions. Par exemple, dans une ligne droite la voiture peut décider de tourner alors qu'elle aurait dû aller tout droit. Dans ce cas-là la décision est mauvaise, mais c'est plus difficile de l'identifier, car elle ne sort pas de la piste.

Pour la vitesse, je choisis trois niveaux de granularité pour être sûr de bien pouvoir accélérer et ralentir. **Figure 5**

Figure 5



Création d'un modèle

Pour créer un nouveau modèle, tout se passe dans la console AWS, dans le service DeepRacer. La première étape consiste à donner un nom au modèle et ajouter une description. La deuxième étape est la sélection du circuit, parmi les 23 disponibles.

Je commence avec un entraînement d'une heure sur une piste assez simple comme re:Invent 2018 ou le London Loop Training. Ensuite, je clone mon modèle sur des pistes plus techniques avec plus de virages et de décors comme Stratus Loop.

La troisième étape est le choix du type de course ainsi que les caractéristiques de la voiture. Dans notre cas nous choisirons le Time Trial avec la voiture créée à l'étape précédente. Pour la dernière étape, il faut créer sa fonction de récompense, sélectionner la valeur des hyperparamètres et de la durée d'entraînement et c'est parti pour l'entraînement ! **Figure 6**

Les fonctions de récompense

La fonction de récompense prend en entrée plusieurs données sur l'environnement et la voiture pour en déduire un score qui correspond à la récompense. Plus la récompense est haute, plus la décision qu'a prise l'agent est bonne. Plus la récompense est basse, plus la décision est mauvaise.

Pour créer notre fonction de récompense, nous avons plusieurs variables à notre disposition :

- all_wheels_on_track, boolean : indique si toutes les roues sont toutes sur la piste
- x, y, [float, float]: les coordonnées de la voiture sur la piste en mètres
- closest_waypoints, [int, int]: les deux points de repère les plus proches
- distance_from_center, int : la distance entre la voiture et le centre de la piste
- is_crashed, boolean : indique si l'agent s'est crashé
- is_left_of_center, boolean : indique si l'agent est à gauche de la piste
- is_offtrack, boolean : si l'agent est en dehors de la piste
- is_reversed, boolean : si la voiture est en sens inverse
- heading, float : le niveau de rotation de la voiture (lacet)
- progresse, float : le pourcentage de la piste parcourue
- speed, float : la vitesse de la voiture en m/s
- steering_angle, float : l'angle de braquage de la voiture
- steps, int : nombre d'expériences effectuées
- track_length, float : longueur de la piste en mètres

- track_width, float : largeur de la piste en mètres
- waypoints, [(float, float)] : liste des coordonnées des waypoints

Le code ci-dessous nous montre deux fonctions de récompense de base : reward_function_heading et reward_function_distance_from_center.

```
def reward_function_from_center(params: dict) -> float:
    """
    Example of reward function using distance from center
    :param params: All parameters of the environment in a dictionary
    :return: the reward
    """

    # Read Input variable
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']

    # Penalize if the car is to far away from the center of the track
    marker_1 = 0.1 * track_width
    marker_2 = 0.5 * track_width

    if distance_from_center <= marker_1:
        reward = 1
    if distance_from_center <= marker_2:
        reward = 0.5
    else:
        reward = 1e-3

    return float(reward)
```

```
def reward_function_heading(params: dict) -> float:
    """
    Example of reward function using waypoints and heading to make the car point in the right direction
    :param params: All parameters of the environment in a dictionary
    :return: the reward
    """

    # Read Input variable
    waypoints = params['waypoints']
    closest_waypoints = params['closest_waypoints']
    heading = params['heading']

    # Initialize the reward with a typical value
    reward = 1

    # Calculate the direction of the center line based on the closest waypoints
    next_points = waypoints[closest_waypoints[1]]
    prev_points = waypoints[closest_waypoints[0]]

    # Calculate the direction in radius, arctan2(dy, dx) the result is (-pi, pi) in radians
    track_direction = math.atan2(next_points[1] - prev_points[1], next_points[0] - prev_points[0])

    # Convert to degree
    track_direction = math.degrees(track_direction)
```

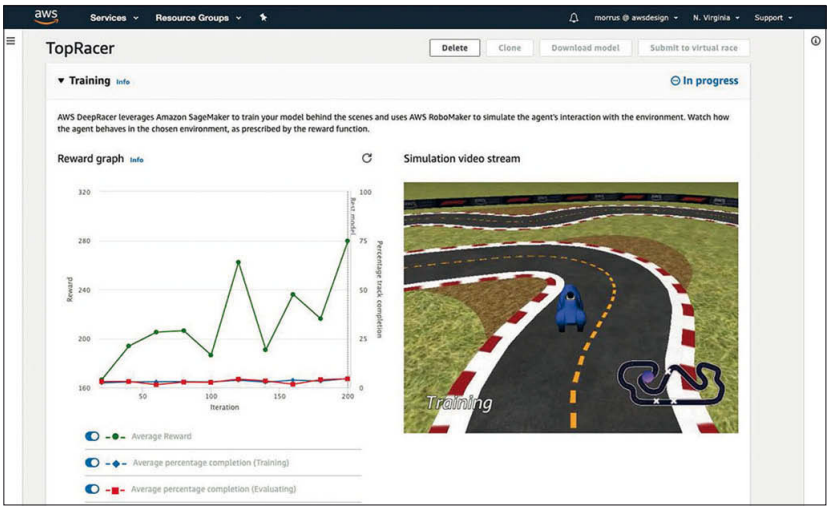


Figure 6

```
    # Calculate the difference between the track direction and the heading direction of the car
    direction_diff = abs(track_direction - heading)
    if direction_diff > 180:
        direction_diff = 360 - direction_diff

    # Penalize the reward if the difference is too large
    DIRECTION_THRESHOLD = 10.0
    if direction_diff > DIRECTION_THRESHOLD:
        reward *= 0.5

    return float(reward)
```

La fonction de gauche, reward_function_heading est très simple. Pour chaque expérience, plus la voiture est proche du centre de la piste plus elle est récompensée. L'avantage de cette fonction est sa simplicité, ce qui rend l'entraînement très rapide. En revanche, ce type de fonction n'incite pas la voiture à aller vite, car la vitesse n'est pas récompensée et la qualité de la trajectoire non plus.

La fonction de droite est un peu plus évoluée. Elle utilise les waypoints et le heading pour vérifier que la voiture se dirige dans la bonne direction. Pour cela, elle prend les coordonnées du waypoint précédent, du suivant et de la voiture puis calcule l'angle entre ces trois points. Si l'angle est supérieur à un seuil, alors la voiture est pénalisée.

Un des gros avantages de cette fonction est le fait qu'elle pénalise la voiture lorsque sa trajectoire oscille trop. Dans une course physique, avoir un modèle qui oscille fait sortir la voiture de la piste.

En revanche, le problème de ce modèle est qu'encore une fois, il ne récompense pas la vitesse. En plus de cela, utiliser un seuil rend l'entraînement plus complexe. Il faut mieux utiliser une récompense continue (exemple : reward = direction_diff * coefficient).

Notre fonction de récompense

J'ai participé à l'édition 2019 avec Arthur Pace et Antony Libault, lorsque nous travaillions dans l'AIE de Capgemini. Nous avons remporté la course en effectuant un tour de 13 secondes. Nous avons ainsi pu nous qualifier pour la finale mondiale qui a eu lieu à Las Vegas en décembre 2019.

Alors comment avons-nous remporté cette course ? Avec beaucoup de patience, de café, d'itérations, d'échecs et d'apprentissage, mais surtout avec une fonction de récompense très efficace.

Une des principales caractéristiques de notre approche est le fait que nous utilisons des variables globales pour stocker des informations pendant toute la durée de l'entraînement. De ce que nous avons pu observer chez nos concurrents, nous étions parmi les seuls à avoir cette approche.

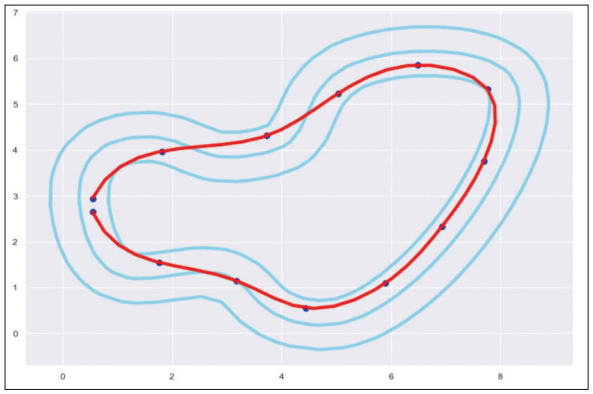
Cela nous a permis par exemple de stocker la position de la voiture à chaque instant pour analyser sa trajectoire et calculer l'angle qu'elle fait entre sa position précédente sa position actuelle et le centre de la piste.

Nous nous sommes également inspirés de différentes sources notamment un papier de l'INRIA qui nous a beaucoup inspirés : ***End-to-End Race Driving with Deep Reinforcement Learning*** par Maximilian Jaritz, Raoul de Charette, Marin Toromanoff, Etienne Perot, Fawzi Nashashibi (https://team.inria.fr/rits/files/2018/02/ICRA18_EndToEndDriving_CameraReady.pdf).

Nous avons entraîné le modèle pendant 7 heures, en utilisant Huber et un petit learning rate avec peu d'épisodes entre chaque itération.

```
@staticmethod
def reward_function(
    distance_from_center: float,
    throttle: float,
    total_progress: float,
    alpha: float,
) -> float:
    """
    Function used to compute the reward.
    :param distance_from_center: how far is the car from the center of the track
    :param throttle: the speed of the car
    :param total_progress: % of progress made by the car on track
    :param alpha: the angle made by the car between its current position,
the previous one and the center of the track
    :return: the reward
    """

    MAX_SPEED = 3.2
    SPEED_FACTOR = 0.9
    DISTANCE_CENTER_COEFFICIENT = 4.5

    reward = 1e-5
    is_speed = 0

    # Reward depending of far is the car from the center
    is_centered = DISTANCE_CENTER_COEFFICIENT * (1 - np.exp(np.abs
(distance_from_center)))

    # Reward if the reward is over a threshold
    if throttle > SPEED_FACTOR * MAX_SPEED:
        is_speed = 1

    # Reward depending on alpha angle
    reward_cos = throttle * (np.cos(alpha) - np.max([1 - total_progress
/100 - 0.2, 0]))

    reward += reward_cos + is_centered + is_speed

    return float(reward)
```

Figure 7



La finale à Las Vegas

La finale s'est déroulée au MGM Grant de Las Vegas. Nous avons travaillé pendant plusieurs mois sur un nouveau modèle avec une approche complètement différente. Nous sommes partis du principe que pour être les plus rapides sur la piste, la voiture devait suivre la trajectoire parfaite.

Pour cela, avons développé un module en Python qui calcule la meilleure trajectoire possible pour un jeu de waypoints en fonction d'une approximation des caractéristiques de la voiture.

Pour cela, nous avons utilisé l'algorithme du one star pour trouver le chemin le plus court sur la piste, puis l'algorithme du rear wheel feedback pour optimiser les trajectoires de la voiture. Pour plus de détails : https://github.com/matrousseau/AWS-Deepracer-Optimal-Path-Generator

Une fois les waypoints générés, il n'y a plus qu'à les importer et récompenser la voiture lorsqu'elle suit la trajectoire parfaite.

Malheureusement, cette approche n'a pas bien fonctionné, car le modèle avait des défauts et nous n'avons pas été en mesure de le tester sur une vraie piste avant le jour J. Cela ne nous a pas empêchés de faire quelques beaux tours et passer très proches de la qualification en finale. **Figure 7**

Pour aller plus loin

Pour pouvez rejoindre la communauté DeepRacer sur Slack qui est très active : deepracer.io, ou sur YouTube où des gagnants du monde entier expliquent leur approche : AWS DeepRacer Community. Il existe aussi de nombreuses façons d'analyser ses entraînements. Un challenge a même été lancé pour récompenser la meilleure façon de les analyser : https://blog.deepracing.io/2019/10/24/aws-deepracer-community-log-analysis-challenge/

Enfin, je vous invite grandement à regarder la finale remportée par Sola avec un modèle extrêmement impressionnant.

Série IoT : Welcome to the real world”

PARTIE 4

Implémentation d'une solution IoT - approche PaaS

Jon Mikel Inza

Solutions Architect IoT chez Codit

De formation technique mais avec un parcours varié, je travaille aujourd'hui en tant que Solutions Architect chez Codit France. Mon rôle principal est d'aider nos clients à construire des solutions cohérentes et optimisées par rapport à leurs besoins et objectifs.

Dans les parties précédentes, nous avons décrit une partie de l'univers IoT et les différentes approches (SaaS / PaaS) pour aborder la création de solutions IoT. Cet article complète les publications précédentes en décrivant la mise en place d'une solution orientée PaaS.

Cible fonctionnelle

Solution IoT permettant de :

- Déclencher des « builds » depuis un objet connecté de type bouton.
- Suivre les déclenchements des builds.
- Gérer les boutons (activer, désactiver, statut, etc).

Techniquement parlant, la solution devra permettre de :

- Gérer des devices
- Provisioning
- Gestion des statuts des devices
- Mise à jour des configurations à distance
- Recevoir et intégrer les messages envoyés par les devices (D2C)
- Valider les messages reçus (format et domaines)
- Stocker les messages reçus pour des traitements ultérieurs

Note

Le firmware du bouton choisi pour la série d'articles (Seeed ReButton, cf articles précédents) ne permet d'envoyer qu'un seul type de messages.

Par ailleurs, en fonction de la version du firmware installé dans le ReButton, le device ne pourra pas traiter certains des flux C2D :

- Appel de Direct Methods
- Réception de messages depuis la solution

Pensez donc à faire la mise à jour du firmware si vous souhaitez implémenter ces fonctionnalités en utilisant ce bouton.

Architecture

L'architecture générale de la solution repose sur deux modules :

- IoT Management, qui couvre toute la gestion IoT
- Data processing, qui couvre le traitement et le stockage de la data

Module IoT Management

Le module IoT Management est le module qui prend en charge la gestion des devices et l'ingestion des données brutes. Il s'agit d'un module fondamental dans toute solution IoT.

Vous trouverez sur la **Figure 1** une proposition d'architecture détaillée pour un tel module (adapté aux besoins de l'article).

Les services implémentés sont :

- Azure IoT Hub avec les fonctionnalités de routing ;
- Azure Event Hub ;
- Azure Function ;
- Azure Service Bus ;
- Azure Storage ;
- Azure Service App ;

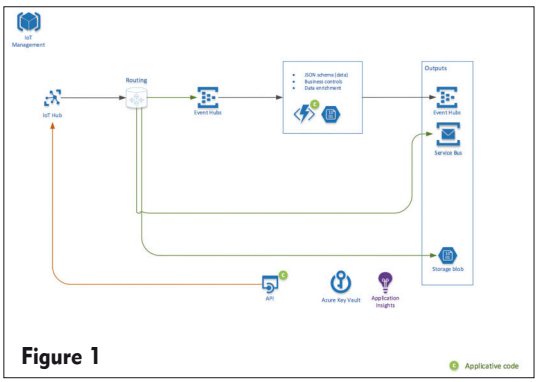


Figure 1

- Azure Key Vault ;
- Azure Application Insights.

L'ensemble des services sont créés au sein d'un même Azure Resource Group, qui est dédié au module.

Azure IoT Hub est le point d'entrée des flux data de la solution. Nous avons choisi de ne pas implémenter un DPS (Device Provisioning Service). L'IoT Hub alimente différents outputs. L'un d'entre eux est un Event Hub qui permet d'adresser une très haute volumétrie de messages (la scalabilité est importante dans les projets IoT). Cet Event Hub alimente une Azure Function qui pour chaque message reçu va contrôler son schéma (schéma au sens JSON). Ainsi, seuls les messages conformes aux formats attendus seront acceptés. Cette architecture peut servir également à :

- Enrichir des messages ;
- Rajouter des contrôles ou logiques métier ;
- Rajouter des mécanismes de processing des informations ;
- Rajouter des mécanismes de transformation de messages ;
- Rajouter des mécanismes riches de routage (ex. : alimentation de bases de données temporelles).

Le Service App contient une API permettant d'interagir avec l'IoT Hub via une « interface » contrôlée et sans adhérence forte. Cette API REST permet de :

- Rajouter un device ;
- Récupérer ses clés ou chaînes de connexion ;
- Désactiver un device ;
- Supprimer un device ;
- Mettre à jour le Twin d'un device (Tags ou Desired properties) ;
- Lancer des jobs (au sens IoT Hub).

Pour terminer, des services techniques comme Azure Key Vault et Application Insights ont été prévus dans le module. Le premier renforce la sécurité en centralisant les secrets et leur gestion. Le deuxième centralise quant à lui l'ensemble des logs du module.

Module Data Processing

Le module Data Processing prend en charge le traitement de premier degré des données brutes qui arrivent au module. Ce

dernier s'occupe aussi d'alimenter les différents modèles de stockage qui serviront d'output du module.
Après l'IoT Management, il s'agit sans doute d'un autre module très important dans les solutions IoT. Il peut être implémenté en suivant différentes variantes. Vous trouverez sur la **Figure 2** celle qui est proposée pour l'article.
L'input de ce module est l'Event Hub du module IoT Management contenant les données brutes envoyées par les devices. Nous savons déjà que tous les messages arrivant à ce niveau respectent les formats attendus. Nous n'aurons donc pas à mettre en place des mécanismes de contrôle de format.
Le module implémente les services suivants :

- Azure Stream Analytics ;
- Azure Event Hub ;
- Azure CosmosDB ;
- Azure Storage ;
- Azure Service App.

L'ensemble des services sont créés au sein d'un même Azure Resource Group dédié au module.
Stream Analytics prend en entrée de fortes volumétries de données (par l'Event Hub) et intègre la capacité de les traiter en streaming. Les cas d'utilisation peuvent être divers (ex : windowing / compressions temporelles, changer le format de données, précalculer des champs, filtrer des données en fonction de règles métier ou des valeurs précalculées, construire des routes sur la base de calcul métier, calculs analytiques, détection d'anomalies, etc).

Note
Il est possible d'implémenter des modules Azure Stream Analytics pour Azure IoT Edge.

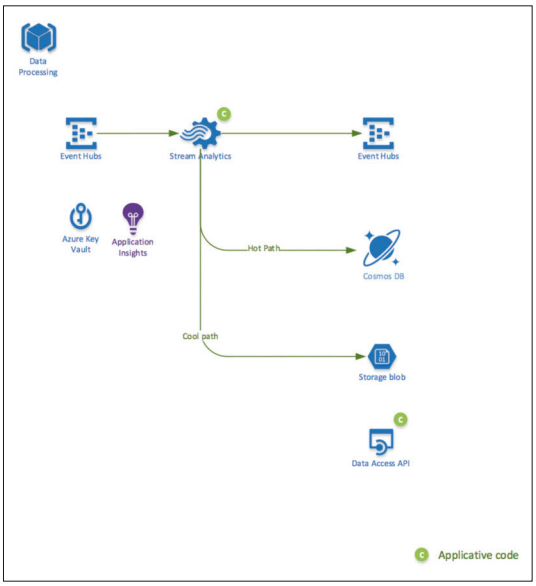


Figure 2

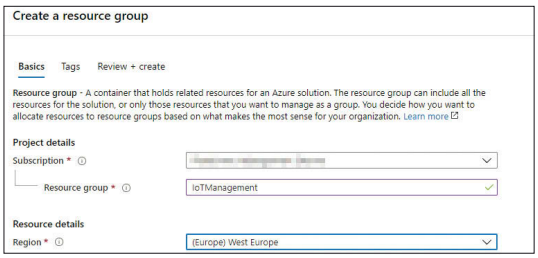


Figure 3

Les outputs de Stream Analytics peuvent être de différents types. Ceux que nous allons retenir dans l'article sont Azure Storage, Azure CosmosDB et Azure Event Hub.
Azure Storage (Blob) permet de stocker les données à froid tandis que CosmosDB les persiste à chaud. Les possibilités de requêtage, coûts et temps de réponse seront ainsi optimisés.
Un autre Event Hub est utilisé pour proposer à d'autres modules les données brutes ou partiellement enrichies.
De même que pour le module IoT Management, nous avons prévu une API REST sous forme de Service App permettant de mettre en place des API d'accès à certaines des données (le code de cette application ne sera pas couvert dans cet article).

Réalisation

Mise en place de la solution

Aujourd'hui, Azure permet différentes façons d'interagir et de gérer la plateforme (portail, Azure Powershell, Azure CLI, ARM, etc.). Nous resterons sur une approche UI pour l'aspect pédagogique et visuel de l'article.

IoT Management

Commençons par implémenter le Resource Group et les services du module IoT Management. **Figure 3**
Après la création du Resource Group, nous pouvons passer à la création de l'IoT Hub.

IoT Hub (Figure 4)

Comme vous pouvez le constater, il est possible d'utiliser une version F1 gratuite pour vos tests et développements (un IoT Hub gratuit par abonnement Azure).
Ensuite, nous devons configurer les routes prévues dans le module. Ces routes ont besoin des endpoints suivants :

- Azure Event Hub
- Azure Storage
- Azure Service Bus (optionnel pour la fin de l'article)

Ceci se fait via la section « Message Routing » du menu de l'IoT Hub. **Figure 5**
La création des routes se fait depuis l'onglet « Routes ».
Figure 6

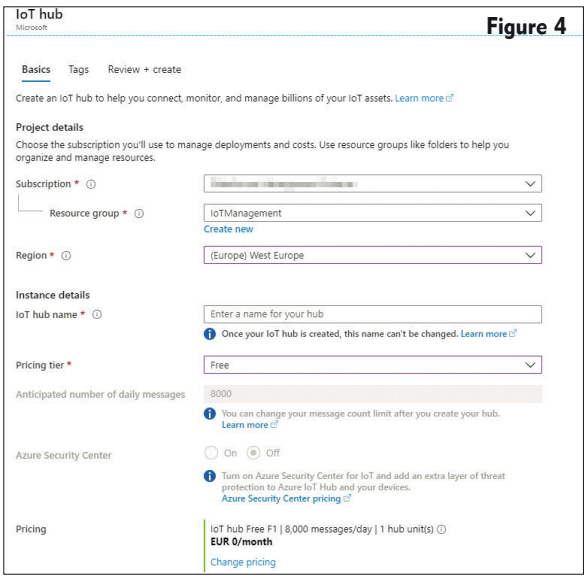


Figure 4

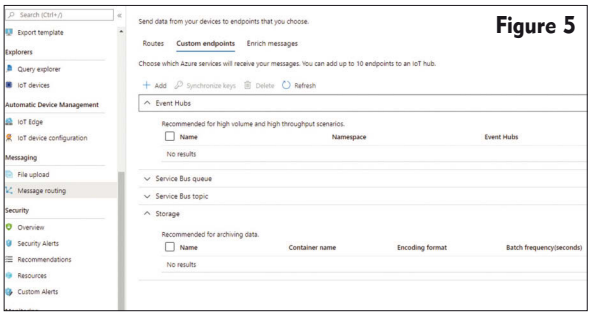


Figure 5

Figure 6 et 11

Si les « Custom Endpoints » n'existent pas au moment où vous souhaitez rajouter les Routes, vous pouvez les créer directement depuis l'assistant de configuration des « Routes ».
Si au contraire, vous préférez créer les services avant les routes, vous aurez besoin de créer :

- Un Event Hub pour les télémétries
- Un Azure Storage

Event Hub

Depuis le portail Azure, ajouter un nouveau service de type Event Hub. L'assistant de création vous demandera les informations pour créer un Namespace pour cet Event Hub.
Figure 7
Une fois le namespace créé, vous pourrez rajouter les instances d'Event Hubs. **Figure 8 et 9**
Nous laisserons les settings par défaut.
Dans ce module, nous aurons besoin de deux Event Hubs :

- « telemetry », recevant les messages routés par l'IoT Hub ;
- « validatedtelemetry », recevant les messages validés (cf. Azure Function – Contrôle des schémas JSON des messages entrants).

Azure Storage

Depuis le portail Azure, ajoutez un nouveau service de type Storage Account. **Figure 10**
Ensuite, allez sur la section « Containers » et créez les containers suivants :

- « validatedtelemetry », permettant de stocker les télémétries à froid ;
- « schemas » (optionnel), si vous souhaitez utiliser le Storage Azure pour garder les schémas de validation des messages entrants (cf. Azure Function – Contrôle des schémas JSON des messages entrants).

Routes

En revenant à l'IoT Hub, pour router les messages de télémétrie envoyés par le device, il est nécessaire de configurer la

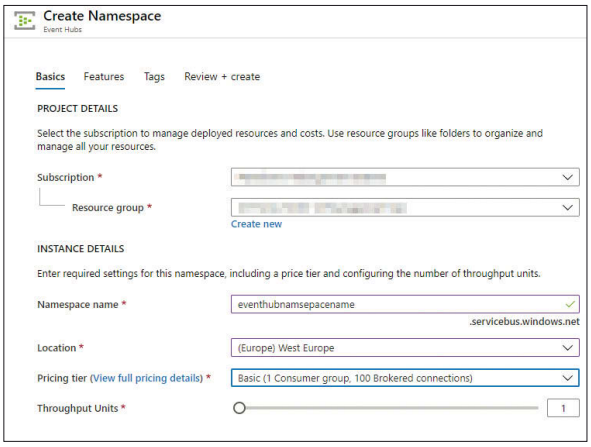


Figure 7

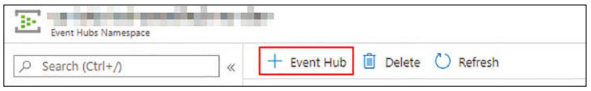


Figure 8

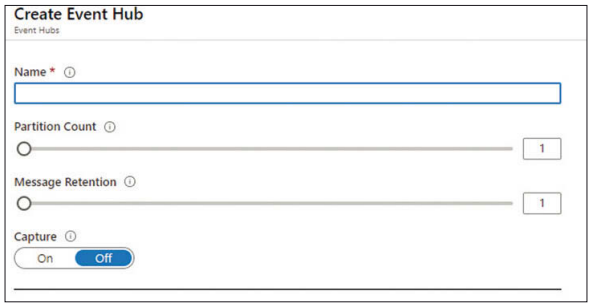


Figure 9

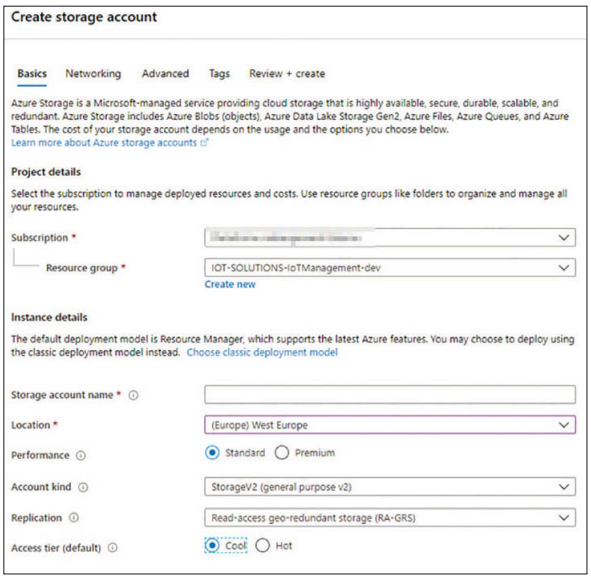


Figure 10

propriété « Data Source » et de choisir « Device Telemetry Messages » dans l'onglet « Routes » du blade « Message routing ».
En ce qui concerne les critères de routage, pour l'instant, nous allons configurer les règles de routage avec un « true ». Cela signifie que tout message en entrée de la route sera routé. **Figure 11**
À ce stade, nous avons les briques principales du module :

- Azure IoT Hub ;
- Endpoints avec les services :
  - Azure Event Hub,
  - Azure Storage.
- Routes.

Il reste l'App Service pour l'API REST, l'Azure Function qui va contrôler les schémas, le KeyVault et l'Application Insight.

Azure Function – Contrôle des schémas JSON des messages entrants

Dans l'architecture proposée, les messages IoT arrivent à l'IoT Hub et celui-ci les route vers les endpoints personnalisés. La télémétrie est orientée vers un Event Hub, qui est l'input de l'Azure Function que nous allons déployer maintenant.

Cette Azure Function a pour but de contrôler le schéma des messages qui lui sont transmis. Il faudra donc :

- La définition des schémas JSON ;
- Écrire le code de l'Azure Function pour réaliser une telle validation ;
- Créer l'Event Hub cible pour les messages valides.

Schémas JSON

Un schéma JSON est un fichier JSON qui définit la structure, les propriétés et les valeurs autorisées de tout message qui le référence. L'idée est la même que pour les XSD avec les fichiers XML.

Exemple de schéma JSON pour les messages envoyés par les boutons ReButton :

Code complet sur www.programmez.com / GitHub

Sources :

https://github.com/jonmikeli/programmez/tree/master/iot-azure-paas/schemas/reButton

Code C#

Nous pouvons désormais écrire une Azure Function utilisant ces schémas pour valider les inputs.

Pour cela, nous créons un projet de type Azure Function (Visual Studio ou Visual Studio Code).

Nous créons ensuite un dossier *SchemaCompareFunction* à l'intérieur duquel nous rajoutons la classe SchemaController.cs avec le code suivant.

Code complet sur www.programmez.com / GitHub

Note

Les clés de configuration sont déjà positionnées pour prendre en compte un certain nombre de paramètres de configuration. Les scripts IaC fournis (cf. plus loin dans l'article) prennent en compte ces paramètres. Ce qui veut dire que si vous utilisez les scripts fournis, vous n'aurez pas à alimenter ces configurations. Par contre, si vous personnalisez ces clés, n'oubliez pas de les mettre à jour dans l'Azure Function.

Vous aurez surement constaté que le code contient deux « outputs » : un Event Hub et un Storage. Le Storage peut ne pas être nécessaire, mais permet de garder une trace dans le « flux » des messages. La méthode qui réalise la validation du schéma est la suivante :

Code complet sur www.programmez.com / GitHub

Une fois les tests faits (oui, il FAUT tester), vous pouvez déployer l'Azure Function.

Code source complet de l'Azure Function :

https://github.com/jonmikeli/programmez/tree/master/iot-azure-paas/sources/apps/IoTManagement.AF

Note

N'oubliez pas d'uploader le schéma vers un stockage accessible et de rajouter la propriété « $schema » aux messages envoyés pour que le schéma soit pris en compte.

Exemple :

```
{
 "$schema": "https://cuiotsiotstoragehubdev.blob.core.windows.net/schemas/rebutton/v1/reButtonBodySchema.json",
 "actionNum": "1",
 "message": "Single click",
 "batteryVoltage": 2.808425
}
```

REST API – Exposition des fonctionnalités de l'IoT Hub

Cette API a uniquement pour but d'intégrer l'IoT Hub avec d'autres systèmes par le biais d'une API REST.

Celle-ci expose certaines des fonctionnalités de l'IoT Hub.

Exemple de fonctionnalités de l'API :

- Provisionning d'un device ;
- Changement de statuts ;
- Mise à jour de Tags ;
- Lancement de jobs IoT Hub ;
- Messages C2D ;
- Direct methods ;
- Twin Desired Properties.
- Etc.

Le détail complet de l'API est exposé via l'OpenAPI (anciennement Swagger). Le service REST repose sur .NET Core 3.1 (C#) (au moment où cet article sera publié, .NET 5 sera en GA et pourrait être utilisé). Cette API n'est pas indispensable pour notre projet. Nous en parlons pour illustrer le type d'architecture qui serait proposée dans un projet réel. Si vous souhaitez déployer l'API dans votre solution, voici le lien vers les sources :

https://github.com/jonmikeli/programmez/tree/master/iot-azure-paas/sources/apps/IoTManagement.API

Data Processing

De même que pour le module IoT Management, nous commençons par créer le Resource Group.

Une fois le Resource Group prêt, nous enchaînons avec le service principal du module : Azure Stream Analytics.

Ce service prend en entrée les messages qui sont délivrés par l'output (Event Hub) du module IoT Management. Stream Analytics permet de traiter une haute volumétrie de messages, en gérant le partitionnement des inputs/outputs et avec, entre autres, la possibilité d'intégrer des fonctionnalités de windowing (calculs, projections sur différents mécanismes de segmentations temporelles).

Les requêtes Stream Analytics permettent aussi d'alimenter différents outputs selon des critères variés, à partir d'un même input.

Figure 12

La prochane étape consiste à créer et configurer les Inputs et Outputs. Ceci se passe dans les blades suivants : **Figure 13**

Input :

- Event Hub (celui contenant les télémétries de sortie dans le module IoT Management).

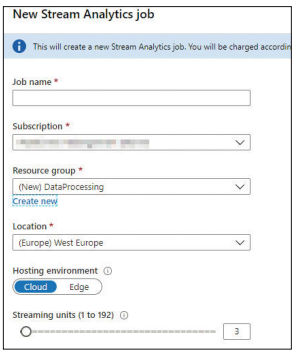


Figure 12

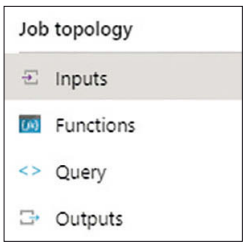


Figure 13

Outputs :
• Event Hub ;
• Storage ;
• Cosmos DB.
Une fois les inputs et outputs définis, c'est au tour de la requête. **Figure 14**
Les requêtes s'écrivent de la manière suivante :

```
SELECT
  *
INTO
  [YourOutputAlias]
FROM
  [YourInputAlias]
```

Le langage ressemble au T-SQL et permet d'utiliser différents types de clauses :
• WHERE ;
• FONCTIONS ;
• WINDOWING/PROJECTIONS.

Note
L'output de type CosmosDB requiert un champ « id ». Il faut veiller à ce que votre requête contienne cette propriété. Sinon, le job Stream Analytics générera une erreur.

Exemple de requête si vous souhaitez générer un « id » random :

```
SELECT
  *,
  GetMetadataPropertyValue([IoTManagement-EventHub-Telemetry], 'EventId') as id
INTO
  [DataProcessing-CosmosDB]
FROM
  [IoTManagement-EventHub-Telemetry]
```

Nous appliquerons le même principe avec le champ « deviceid » (requis par l'output de type Azure Storage) dans la requête alimentant l'output de type Azure Storage. Le but est de structurer le stockage par device, ce qui finit par être très pratique.

Note
Toujours dans le contexte IoT, il est possible d'intégrer des modules Stream Analytics dans Azure IoT Edge. Il existe des cas d'utilisation où ceci génère une valeur importante (ex. : détection d'anomalies à proximité de la source de données).

Synthèse
A ce stade, si nous envoyons des messages conformes au schéma créé, ils devraient :
• traverser l'IoT Hub
• arriver à l'Event Hub
• être validés par l'Azure Function pour être déposés dans un autre Event Hub (en même temps qu'un autre storage de contrôle, cf code)
• rentrer dans le job Stream Analytics pour suivre les flux vers les différents outputs
De même, les messages ne respectant pas le format indiqué ne devraient pas être traités.

Testons l'ensemble

Pour tester l'ensemble des éléments mis en place, nous pouvons utiliser un bouton ReButton ou un simulateur de devices IoT.
Vu que dans l'article précédent nous avons déjà utilisé un bouton physique, nous allons présenter ici l'utilisation d'un simulateur.

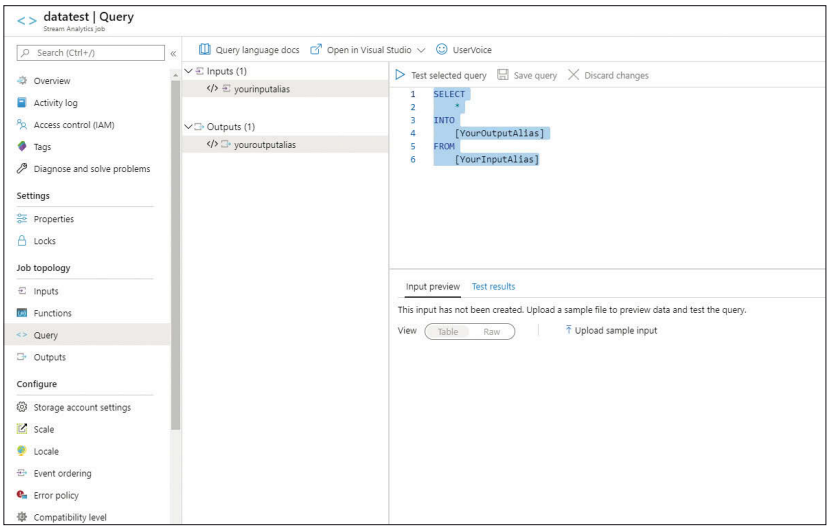


Figure 14

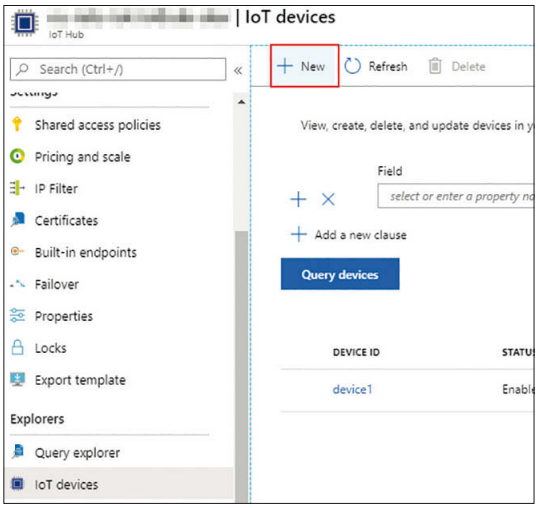


Figure 15

Création d'un device IoT dans l'IoT Hub

Pour passer à la suite et lancer les tests, il est nécessaire de créer un device dans l'IoT Hub (provisionning). Nous aurons besoin de son identifiant et de sa chaîne de connexion afin de pouvoir configurer le matériel (ou les simulateurs).

Création d'un device

L'une des façons de créer un device est en passant par le portail Azure et en allant sur l'IoT Hub. Une fois dans l'IoT Hub, vous trouverez un blade « IoT devices » dans « Explorers ». **Figure 15**
Il suffit de cliquer sur « Add ». **Figure 16**
Il est nécessaire de saisir le Device ID. Vous pouvez laisser les valeurs par défaut des autres paramètres.
Une fois le device créé, vous devriez voir un écran similaire à la **Figure 17**

Récupération de la chaîne de connexion

En cliquant sur l'objet connecté que vous venez de créer, vous pourrez accéder à l'écran contenant les propriétés et récupérer la chaîne de connexion. **Figure 18**

Note

Si vous avez beaucoup de devices à créer, toutes ces opérations peuvent être faites par du bash (Azure CLI), Powershell ou autres mécanismes. La démarche décrite a surtout un but pédagogique.

Gardez bien cette chaîne de connexion. Elle sera nécessaire dans la suite de l'article.

Création du simulateur

Comme mentionné précédemment, nous utiliserons un simulateur pour faire nos tests.

IMPORTANT

Cette approche est valide pour certaines des phases de développement de la solution. Tout projet IoT requiert d'une phase d'intégration. Des tests avec du vrai matériel doivent être faits à un moment donné.

Code Custom

Vous pouvez créer un petit simulateur personnalisé en créant un exécutable imitant les flux d'un device. Pour cela, nous allons construire une application console en .Net Core 3.1 (C#) sous Visual Studio. Si vous le souhaitez, vous pouvez utiliser d'autres langages (C, Python, Node.js, Java). **Figure 19**

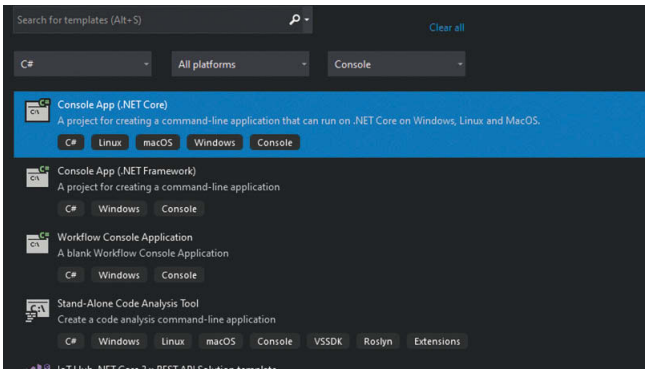
Dans la figure 20, vous saisissez un nom de projet (par exemple, IoT.Simulator) et vous définissez son emplacement dans votre disque.

Une fois le projet créé, vous devez rajouter une référence vers un package NuGet :

Microsoft.Azure.Devices.Client

Ce package intègre toutes les fonctionnalités IoT pour interagir avec IoT Hub depuis un device.

Figure 19



Étape 1 – Création du message métier

Nous créons le message en nous basant sur la description du schéma de la documentation officielle :

```
var reButtonAction = new
{
  actionNum = "1",
  message = "Single click",
  singeClick = "Single click",
  batteryVoltage = 2.8
```

Étape 2 – Création du message IoT

Une fois le message construit, nous le sérialisons et le transformons en message IoT.

```
string jsonData = JsonSerializer.Serialize(reButtonAction);

var message = new Message(Encoding.UTF8.GetBytes(jsonData));

//metadata to define the message type
message.Properties.Add("messageType", "data");

message.ContentType = "application/json";
message.ContentEncoding = "utf-8";
```

Étape 3- Envoi

Il ne reste plus qu'à créer un client IoT et envoyer le message. L'instanciation d'un device se fait via la commande suivante :

```
DeviceClient deviceClient = DeviceClient.CreateFromConnectionString("TO BE REPLACED")
```

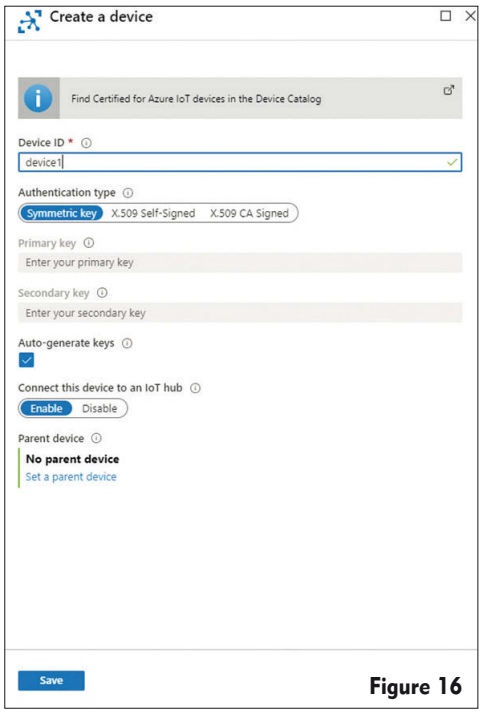


Figure 16

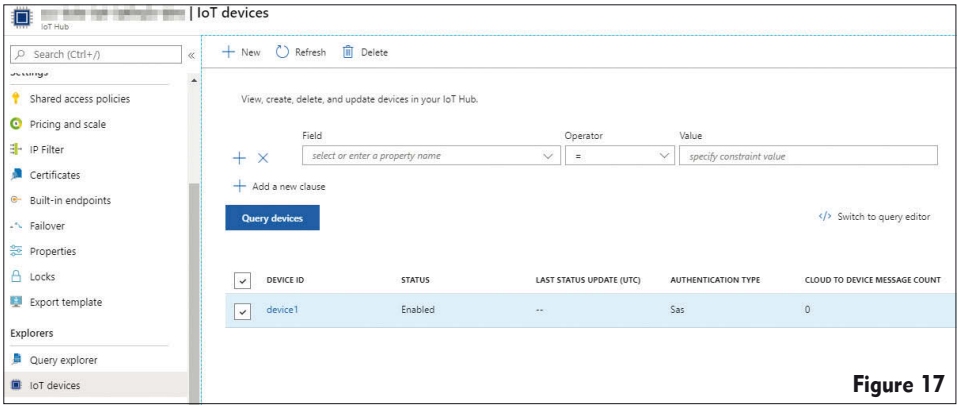


Figure 17

Figure 18

CreateFromConnectionString() attend une chaîne de connexion en paramètre. Vous devez remplacer « TO BE REPLACED » par la chaîne de connexion que vous avez gardée précédemment. L'envoi se fait de la manière suivante :

```
using (DeviceClient deviceClient = DeviceClient.CreateFromConnectionString("TO BE REPLACED"))
{
  await deviceClient.SendEventAsync(message);

  Console.WriteLine("Message envoyé");
  Console.WriteLine(jsonData);
}
```

Code complet de cet exemple :
https://github.com/jonmikeli/programmez/tree/master/iot-azure-paas/sources/simulators/quickstart/IoT.Simulator

Simulateur plus générique

Ce premier simulateur est très simple. Il ne fait qu'envoyer une message IoT vers l'IoT Hub. Or, vous savez désormais que l'IoT va bien au-delà du simple fait d'envoyer/récolter des messages.
Vous trouverez un simulateur plus riche dans le repository ci-dessous : https://github.com/jonmikeli/programmez/tree/master/iot-azure-paas/sources/simulators/advanced/IoT.Simulator
Si vous souhaitez aller encore plus loin avec un simulateur encore plus riche et générique, regardez le lien suivant :
https://github.com/jonmikeli/azureiotdevicesimulator3
Il permet de simuler tous les flux D2C et C2D de manière autonome et intègre différents types de messages.

Autres

Il est également possible d'utiliser Visual Studio, Visual Studio Code, le CLI ou IoT Explorer pour des tests et simulation de devices. Ces outils sont très orientés « développement ».

Tests

À ce stade, nous avons une solution IoT et un simulateur prêt à envoyer des messages. Testons quelques-uns des flux.

Message 1 – Single Click

Nous commençons par envoyer un message simple répondant au flux nominal. Single click message :

```
{
 "actionNum": "1",
 "message": "Single click",
 "batteryVoltage": 2.808425
}
```

Vous constaterez qu'en envoyant ce message, vous ne verrez rien en output. Pourquoi ?
La propriété « $schema » n'est pas renseignée. Du coup, même si le message est bien formé, l'Azure Function ne peut pas faire la vérification. Par conséquent, le message sera refusé.
Pour que cela fonctionne, il faut envoyer le message suivant :

```
{
 "$schema": "https://yourazurestorage.blob.core.windows.net/schemas/rebutton/v1/reButton
BodySchema.json",
 "actionNum": "1",
 "message": "Single click",
 "batteryVoltage": 2.808425
}
```

IMPORTANT

Renseignez la propriété « $schema » avec vos propres valeurs.

Points de contrôle

Pour un contrôle exhaustif, nous devrions valider toutes les traces et les outputs du module. Dans cet exemple, nous regarderons uniquement quelques-uns des outputs de chaque module.

Note

Azure IoT Hub permet aussi d'activer des traces distribuées.

IoT Management - **Figure 20**

Comme vous pouvez le constater, l'output de type Storage account contient bien les messages reçus et conformes au schéma JSON.

Data Processing

Les outputs du module IoT Management alimentent l'input du Data Processing.**Figure 21**
Vous pouvez constater également que ces inputs traversent l'ensemble des processus pour arriver dans l'un des outputs du module Data Processing (type Storage encore une fois).

Message 2 – Double Click

Nous pouvons ensuite remplacer le message envoyé par celui-ci. Il simule un double click du bouton.

```
{
 "$schema": "https://yourazurestorage.blob.core.windows.net/schemas/rebutton/v1/re
Button
BodySchema.json",
 "actionNum": "2",
 "message": "Double click",
 "batteryVoltage": 2.808425
}
```

Message 3 – Low battery

Enfin, nous pouvons simuler un message de batterie faible.
Low battery message :

```
{
 "$schema": "https://yourazurestorage.blob.core.windows.net/schemas/rebutton/v1/re
Button
BodySchema.json",
 "actionNum": "1",
 "message": "Single click",
 "batteryVoltage": 0.2
}
```

Nous pourrions enrichir le Use Case en créant une nouvelle route dans l'IoT Hub qui traiterait les messages où la batterie serait trop faible (ex. : batteryVoltage < 0.5).
Vous pourriez envoyer ces messages vers un endpoint de type Service Bus Topic, auquel pourraient être abonnés des mécanismes d'alerte et/ou de notification.

Desired Property – Update

Nous allons changer de scénario maintenant. La solution IoT va envoyer une notification de changement de configuration à un device. IoT Hub permet de faire ceci par les Twin Properties (Desired).

Étape 1 – Démarrer le simulateur

Si vous choisissez le simulateur « avancé » mentionné précédemment, il suffit de l'exécuter en vérifiant que la clé de configuration "enableTwinPropertiesDesiredChangesNotifications » est à true.

Si vous avez codé votre propre simulateur, pensez à vous abonner à l'événement qui permet de gérer les changements des Twin Desired Properties d'IoT Hub.

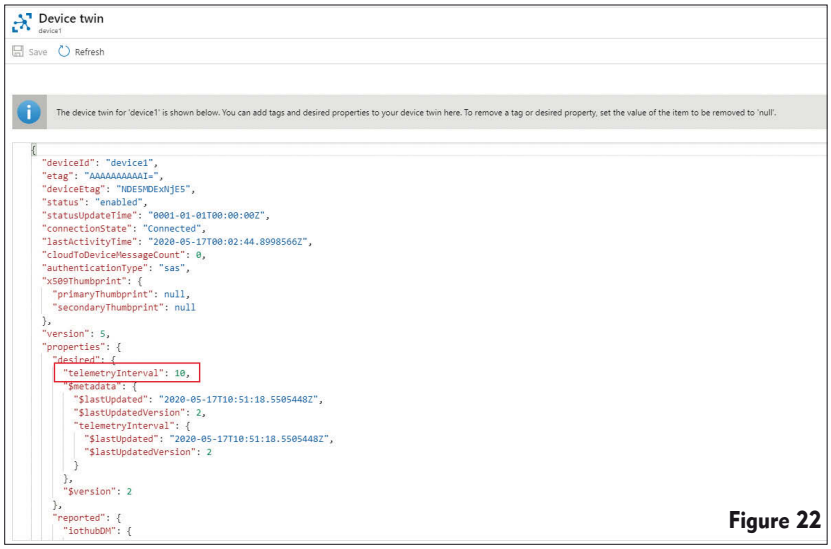


Figure 22

```
await _deviceClient.SetDesiredPropertyUpdateCallbackAsync(OnDesiredPropertyChange,
null);
```

Étape 2 – Envoyer la notification

Les Twin Properties (Desired) peuvent être éditées de plusieurs façons :

- Portail Azure ;
- Azure CLI ou PowerShell ;
- Azure IoT SDK ;
- L'API mentionnée précédemment qui repose sur l'Azure IoT SDK.

Encore une fois, dans cet article, nous garderons une approche UI. Ouvrez le portail Azure au niveau de l'IoT Hub et naviguez à l'explorateur de devices IoT.

Choisissez ensuite le device pour lequel vous souhaitez faire le changement de configuration.

Cliquez ensuite sur « Device twin ». L'écran suivant vous présente l'ensemble du Twin du device en question. **Figure 22**

Ajoutez la propriété « telemetryInterval » au niveau de la section « properties.desired » et cliquez sur «Save ».

Note

Cette propriété n'est pas prise au hasard. Nous l'avons extraite de la documentation du firmware du Rebutton. Pas de magie . **Figure 23**

Comme vous pouvez le constater, le device reçoit la notification de demande de changement de configuration par la solution IoT.

Exécution de commandes

Supposons qu'un nouveau firmware du ReButton permette de gérer davantage de flux C2D.

En utilisant le simulateur fourni, simulons le Reboot du device

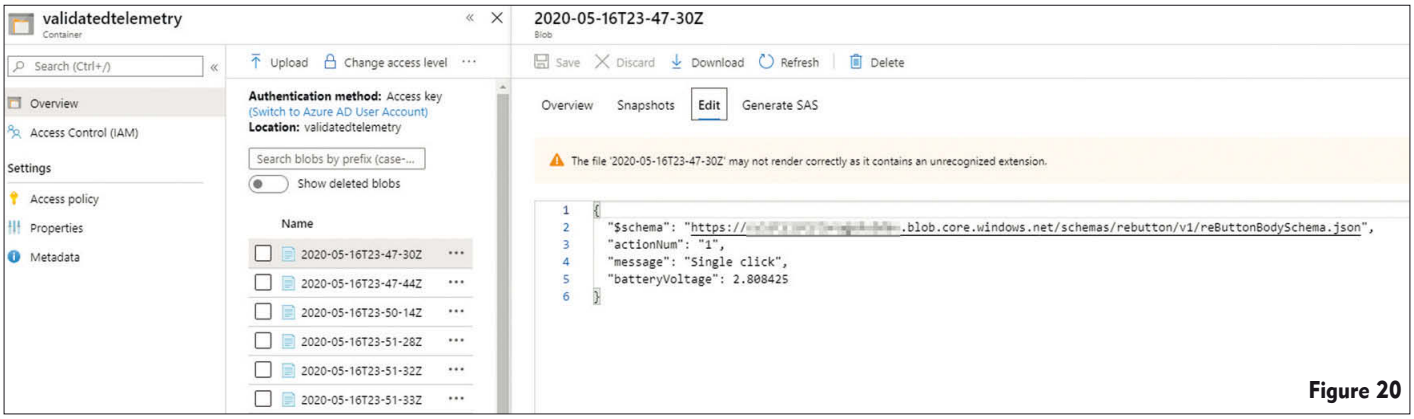


Figure 20

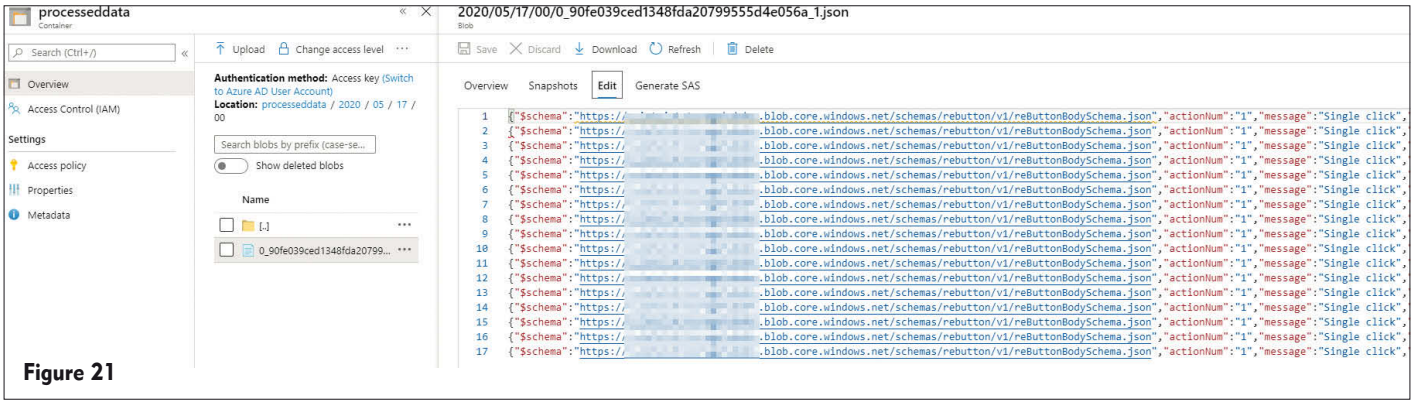


Figure 21

à distance. Ceci peut se faire par les Direct Methods (appel à distance de méthodes dans le device, via l'IoT Hub).
Il ne s'agit pas de se connecter directement au device pour faire un appel.
Cette solution passe par l'IoT Hub qui gère la sécurité, les droits et les canaux de communication avec le device.
Pour appeler une Direct Method, nous devons repartir à la page affichant les propriétés du device (IoT Hub > IoT Devices).

Note

Il faut bien choisir le device qu'on cible et veiller à ce que le simulateur soit démarré. Sinon, comme dans la vie réelle, l'appel sera perdu (les Direct Method sont des mécanismes synchrones).

Figure 24

En choisissant « Direct Method », nous accédons à la page qui nous permet de spécifier le nom de la Direct Method et son payload. Dans notre cas, la Direct Method s'appelle « Reboot » et il n'y a pas de payload. **Figure 25**
Il suffit ensuite de cliquer sur « Invoke Method » (en haut, à gauche). Comme vous pouvez le constater sur la capture du simulateur, l'appel a été bien reçu et le device a simulé le Reboot.**Figure 26**

Conclusion

L'ensemble des articles ont permis d'aller un peu plus loin dans la présentation de l'univers IoT en approfondissant ses raisons d'exister et son expansion dans différents domaines.
Nous avons pu identifier les problématiques les plus communes auxquelles se confrontent les projets IoT. Ces problématiques peuvent être de différents types : techniques, équipe, compétences, business. L'IoT exige la capacité de prendre du recul et analyser les contextes avec une perspective globale. L'adaptation des business models joue un rôle important dans la durée de vie des projets de ce type.
Les compétences restent une problématique centrale mais les progrès techniques et le gain en maturité des solutions absorbent une partie des difficultés. Aujourd'hui, il existe des plateformes technologiques prêtes et matures pour adresser les projets IoT avec sérénité. L'industrialisation, les méthodes de travail, les design patterns et le DevOps rajoutent des mécanismes de sécurisation dans la réalisation de ce type de projets. L'exemple pratique décrivant une solution PaaS s'ajoute à l'approche SaaS présentée antérieurement. L'approche PaaS est sans doute plus complexe, requiert plus de compétences et nécessite un TTM plus important. Néanmoins, elle est plus riche en termes de possibilités (techniques et fonctionnelles), évolutivité, intégrabilité, contrôle et offre un parcours plus long. Il n'y a pas de bonne ou de mauvaise approche. Chaque approche apporte des solutions à des problèmes concrets ; le bon choix commence par l'analyse du contexte et des objectifs du projet. En synthèse, le marché IoT est aujourd'hui technologiquement riche. Il intègre un certain niveau de maturité pour être adressé avec du sens d'un point de vue business et fonctionnel.

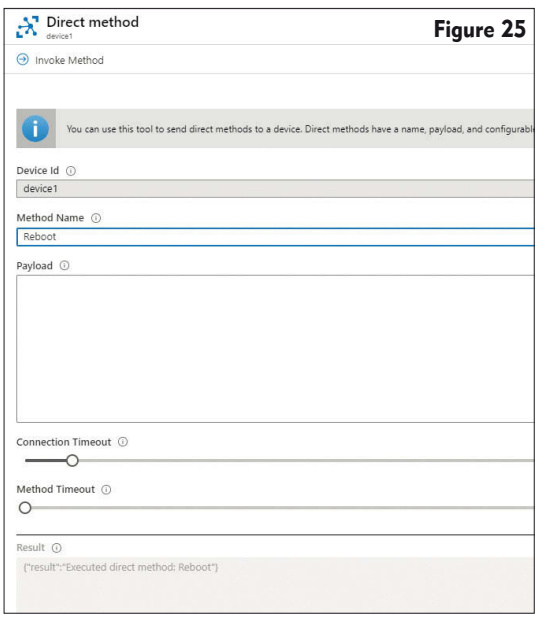
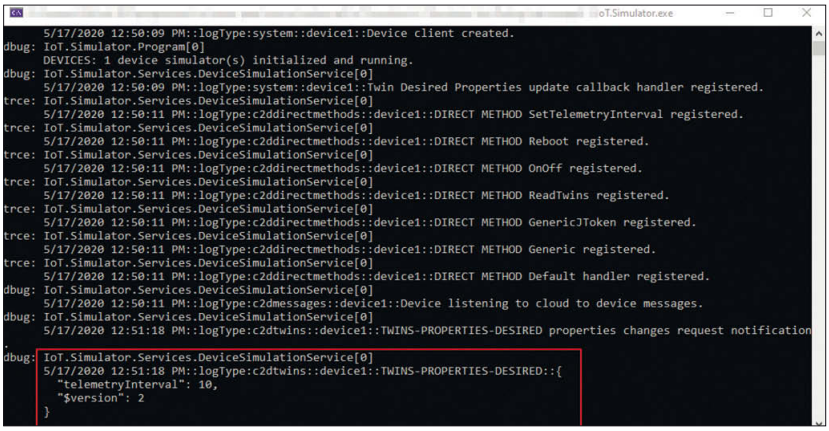


Figure 25

Figure 23



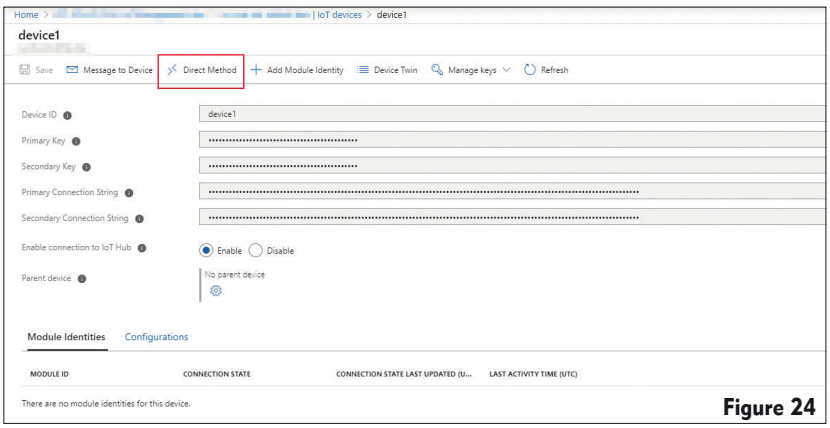


Figure 24

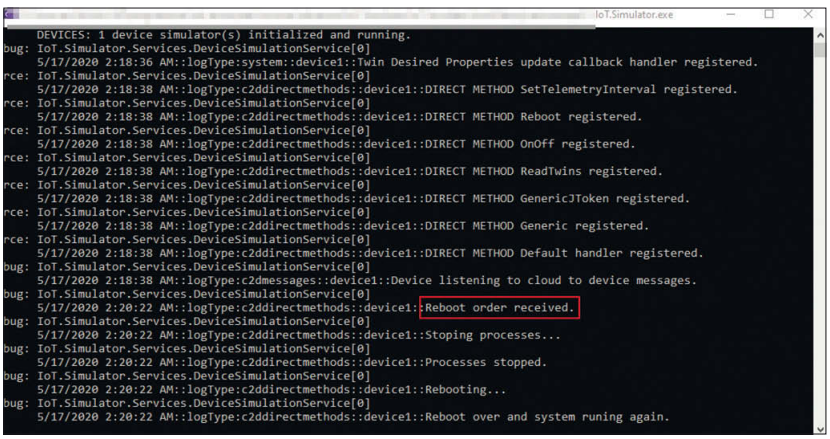


Figure 26

Il ne faut jamais oublier la doc...

LES PROCHAINS NUMÉROS

Programmez! n°247

Spécial Maker / IoT / Coding4fun

Disponible dès le 2 juillet

Hors Série #4 ÉTÉ

100% JAVA

Disponible dès le 9 juillet

PROGRAMMEZ!

Programmez! n°246
mai - juin 2021

Directeur de la publication & rédacteur en chef
François Tonic
ftonic@programmez.com

Contacter la rédaction
redaction@programmez.com

Ont collaboré à ce numéro
La rédaction de ZDnet.fr
Commitstrip

Les contributeurs techniques
Laurent Julliard
Stéphane Ducasse
ZDNet
François Tonic
Coralie Nohel
Arnaud Thiefaine
Laurent Ellerbach
Benoît Prieur
Jean-Christophe Riat
Marwa Thlithi
Abdel-Latif Mabrouck Matroud
Laura Fontaine
Jules Perrodon
Bruno Solforosi
Jérémy Lejeune
Dorra Bartaguiz
Philippe Boulanger
Matthieu Rousseau
Jon Mikel Inza

Couverture
© jemastock

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 €.

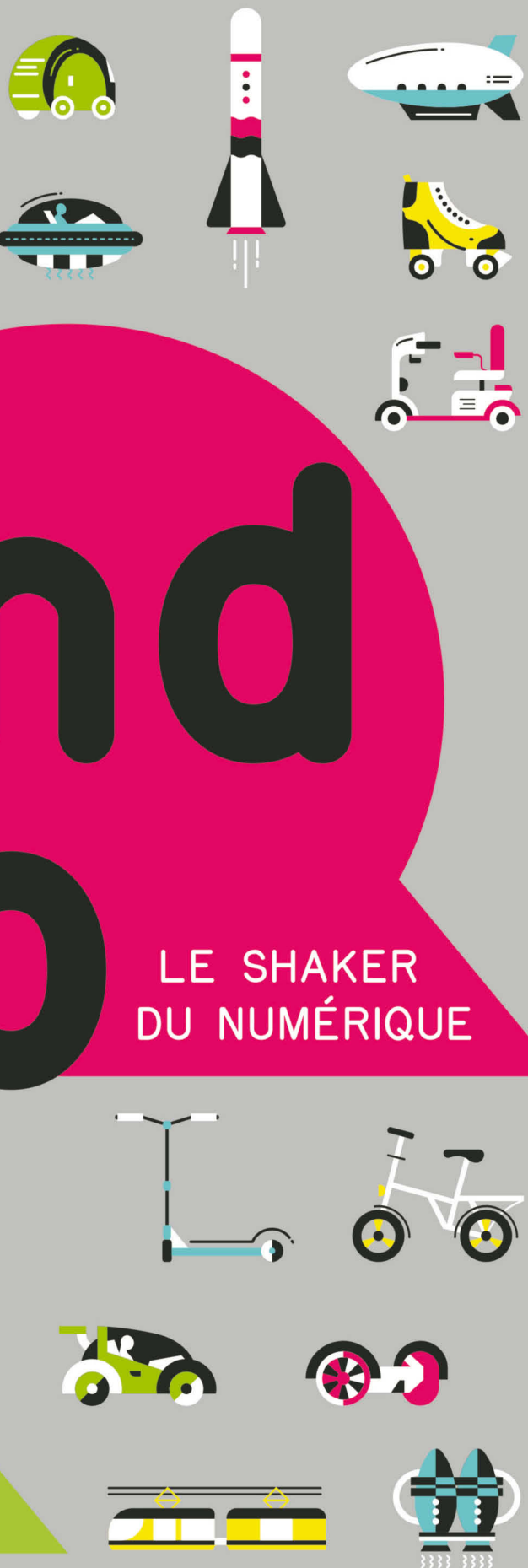
Nefer-IT
57 rue de Gisors, 95300 Pontoise France
redaction@programmez.com
Tél. : 09 86 73 61 08

14_15
juin
2021

CENTRE DES CONGRÈS
DE LYON

blend web mix

LE SHAKER
DU NUMÉRIQUE



LA CUISINE DU WEB

GRANDLYON
la métropole

blendwebmix.com

CRÉATION TANGUY GUÉZO • BUREAU-VIF.COM

NOUVELLE VERSION • WINDEV 26, AGL DEVOPS LOW-CODE

DÉVELOPPEZ UNE FOIS, PUIS COMPILEZ AU CHOIX POUR
WINDOWS, LINUX, MAC, INTERNET, IOS, ANDROID

DÉVELOPPEZ 10 FOIS PLUS VITE

N°1 EN FRANCE

926 NOUVEAUTÉS INCONTOURNABLES

• DEMANDEZ LE DOSSIER COMPLET GRATUIT • TÉLÉCHARGEZ LA VERSION EXPRESS GRATUITE

PCSOFT.FR

TÉL: 04 67 032 032

Logiciel professionnel. Document non contractuel. WINDEV permet de compiler pour Windows et Linux. WEBDEV compile pour Internet. WINDEV Mobile compile pour Mac, iOS et Android.