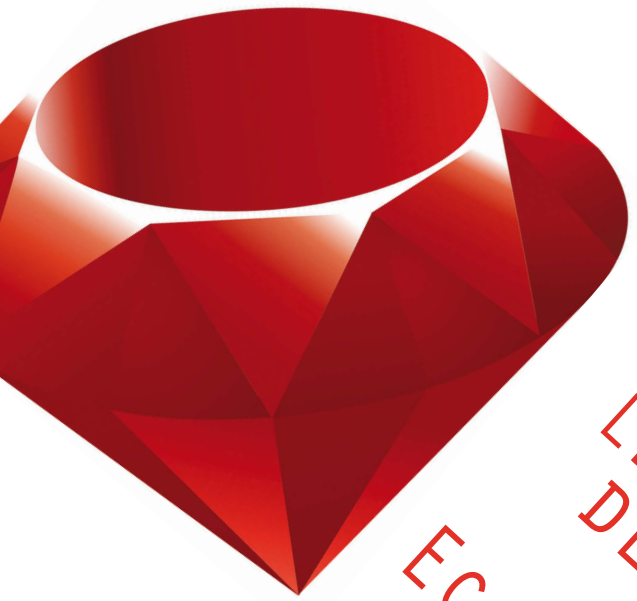
PROGRAMMEZ!

Le magazine des dévs

N°251
03/04
2022



RUBY A 10 ANS

PROGRAMMATION QUANTIQUE

LES NOUVEAUTÉS DE JAVA 18

ECO-CONCEPTION PARTIE2

DOSSIER JAVASCRIPT

DOSSIER FLUTTER

M 04319 - 251 - F: 6,99 € - RD

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

Toujours disponible !

PROGRAMMEZ!

Le magazine des dévs

SPÉCIAL HIVER 2022

100% JEUX

Comment créer un jeu ?

Les outils Les technologies

Devenir développeur de jeux

Consoles open source

M 01642 - 6H - F: 6,99 € - RD

| UN | NUMÉRO | CODÉ | COMPILÉ | PAR | POUR | LES DÉVELOPPEUSES | ET | 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

100 % JEUX

Contenus

Divers

Abonnement numérique (format PDF)
directement sur www.programmez.com

L'abonnement à Programmez! est de 55 € pour 1 an, 90 € pour 2 ans.
Abonnements et boutiques en pages 42-43

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

PROGRAMMEZ!251*

English.content.programmez = show;

En février dernier, le framework .net fêtait son 20e anniversaire. Officiellement, la version 1.0 fut déployée le 13 février 2001. Les premières bêtas remontent à 2001. En 20 ans, le framework a beaucoup évolué et a été remplacé par le nouveau .Net 6 et .Net Core. Au fil des ans, .Net a beaucoup grossi et Microsoft a multiplié les versions, les modèles de développement. Parfois, dans le plus grand désordre.

Aujourd'hui, Microsoft a largement réorganisé .Net pour redonner de la clarté. Xamarin disparait au profit de MAUI qui arrivera au printemps / été. Reste à voir si MAUI qui se veut être le nouveau modèle de développement multiplateforme autour de C# va pouvoir faire bouger le marché. Aujourd'hui, il faut avouer que Flutter occupe largement le terrain avec les frameworks JS. L'intérêt immédiat est qu'un dév .Net / C# trouvera vite ses marques mais pour les autres, l'intérêt semble limité.

Au-delà de Flutter, l'offre multiplateforme déborde. Kotlin propose sa propre approche mais là encore, est-ce véritablement intéressant ? Oui, le langage sort ouvertement du slogan « langage Android » mais le succès auprès des dévs est loin d'être assuré.

Il faut arrêter avec ce que l'on peut appeler le syndrome du framework JavaScript : fais-toi plaisir, sors ton framework JS. Chaque semaine ou presque, il y a une nouvelle lib meilleure que les précédentes. Tout le monde sait que 99 %, disons 95 % pour être sympa, de ces frameworks n'existeront plus d'ici 6 à 12 mois. Notre environnement technique est tellement complexe et encombré, qu'il devient difficile de s'y faire une place. Google a mis des années pour mettre en avant Go et Dart. Et encore, Dart existe via Flutter. Rust fait l'actu grâce à son arrivée officielle dans le noyau Linux.

Sommes-nous condamnés à utiliser toujours les mêmes langages ? Non, pas forcément. Un nouveau langage réussit à s'imposer dans le top 5 tous les 10 ans. Nous avons besoin de stabiliser et de durabilité des choix techniques. Et beaucoup de dévs travaillent toujours des technologies parfois anciennes, voire, dinosaures (Cobolus Rex se porte toujours très bien). Mais le legacy est tellement important qu'il est impossible de tout casser. Même en Java, C#, PHP, C++, l'existant représente des milliards de lignes de code !

Fondamentalement, ce n'est pas le développement qui change mais le métier de dév avec l'agilité, le DevOps, les bonnes pratiques. Mais il faudrait aussi arrêter de vouloir aller toujours plus vite, de réduire les délais de développement, couper les tests.

Un exemple me vient à l'esprit, la nouvelle app de la SNCF. De nombreuses critiques ont été formulées très rapidement notamment sur l'ergonomie parfois déroutante et inutilement complexe. Le dév n'est pas à blâmer car finalement il développe ce qu'on lui dit de développer. Les défauts sont « par design » par contre, dans un processus agile, et particulièrement en craft ou en Lean, les défauts auraient dû être détectés et remontés plus rapidement pour corriger avant le déploiement. Erreurs d'UX et d'XP (oui nous aimons les sigles et abréviations en informatique), bugs, résultats surprenants ou franchement incompréhensibles, mode sombre imposé et impossible à désactiver, etc. Le projet était ambitieux : transformer l'expérience numérique. SNCF Connect a demandé plus de 18 mois de développement en utilisant Flutter, la mobilisation de 200 développeurs. Maintenant, il s'agit de stabiliser l'app, de fixer les bugs et surtout de faire évoluer l'UX. A vouloir rajouter de nouvelles fonctionnalités, de nouveaux transports, de tout fusionner en une seule app, on peut aboutir à une app bancale même si les fondations peuvent être bonnes.

**En réalité n°257*

François Tonic

https://www.youtube.com/channel/UCi80jwLjV9k9ao92r_LGgpg

PROCHAIN NUMÉRO

PROGRAMMEZ! N°252

Disponible le 6 mai 2022

INCONTOURNABLE !

Inside <GIT>

100 % CI/CD
100 % GIT

60 PAGES

Disponible sur programmez.com et en impression à la demande sur Amazon.fr

Sncf connect : un lancement en fanfare

Fin janvier, la SNCF a lancé son nouveau service en ligne : SNCF Connect. Un projet (trop ?) ambitieux visant à unifier les différents services de réservation de billets de la SNCF dans une interface commune, redésignée pour l'occasion. Malheureusement, les premiers utilisateurs de l'application se sont surtout heurtés à des bugs en pagaille, allant de l'absence de certains trajets dans les résultats de recherche à la disparition des dossiers voyageurs. Une situation qui a poussé la SNCF à promettre « des améliorations » pour le service, qui devraient être mises en œuvre au début du mois de mars.

La justice américaine saisit 3,6 milliards en cryptomonnaie

Une saisie record, même pour le monde des cryptomonnaie assez habitué aux vols et autres escroqueries : les autorités américaines ont annoncé l'arrestation d'un couple d'Américains accusés d'avoir blanchi les sommes issues du piratage de la plateforme Bitfinex en 2016. Au total, les forces de police ont mis la main sur la rondelette somme de 119 754 bitcoins, évalués à 3,6 milliards de dollars au moment de la saisie.

Données personnelles : le coup de pression Meta

Un peu plus de 18 mois après l'invalidation du Privacy Shield, les États-Unis et l'Europe ne sont toujours pas parvenus à s'entendre sur un nouvel encadrement des échanges de données personnelles. Et cela n'arrange pas Meta (Facebook), qui envisage dans son rapport annuel la possibilité que Meta se retire du marché européen en l'absence d'accord. « Ce n'est pas une volonté de notre part, simplement un risque que nous envisageons » précisent les porte-paroles de la société américaine dans un post de blog.

Health Data Hub : rien avant les élections

Le secrétaire d'État au numérique Cedric O, s'est exprimé devant le Sénat sur l'avenir du Health Data Hub, le projet gouvernemental visant à proposer une plateforme centralisée pour l'hébergement des données de santé. Initialement hébergé sur le cloud de Microsoft, Azure, le projet avait fait face à de nombreuses critiques avant d'être finalement mis en pause par la CNIL, qui réclamait un appel d'offres pour le choix de l'hébergeur. Comme l'explique le ministre, les grands changements envisagés pour remettre le Health Data Hub dans les clous demandent « plus de sérénité », le sujet étant devenu « excessivement politique », et sont donc renvoyés après l'élection présidentielle.

Google Analytics dans le viseur de la CNIL

La CNIL a annoncé avoir interdit à un gestionnaire de site web non nommé d'utiliser Google Analytics pour les mesures d'audience. Une décision qui se fonde également sur l'absence d'accord encadrant les échanges de données personnelles entre les États-Unis et l'Europe, et l'absence de garanties offertes par Google Analytics pour protéger les données des citoyens européens des outils de surveillance du gouvernement américain. Le groupe visé par la décision a donc un mois pour cesser d'utiliser le service de mesure d'audience, et tous les autres utilisateurs de la solution très populaire de Google ont tout intérêt à envisager des solutions de repli avant que la CNIL s'en mêle.

Publicité ciblée : FloC s'offre un successeur

FloC a fait un flop : la proposition de technologie de publicité ciblée par Google n'a pas survécu aux nombreuses critiques formulées par ses détracteurs. Fidèle à son habitude, Google a donc choisi d'abréger les souffrances du projet et de proposer une nouvelle approche. Celle-ci se nomme « Topics » et s'appuiera sur une API, qui partagera à l'annonceur trois centres d'intérêt définis pour l'utilisateur parmi une liste tirée de son historique de navigation récent. Une alternative qui, Google le promet, sera aussi efficace que la publicité s'appuyant sur les cookies tout en restant respectueuse de la vie privée des utilisateurs.

ARM Nvidia : les plaisanteries les plus courtes sont les meilleures

Cela fait plusieurs mois que la situation s'était compliquée pour la fusion entre Nvidia et ARM. Mais finalement, Nvidia et Softbank, le propriétaire d'ARM, ont décidé de couper court à leur projet de fusion et l'ont annoncé officiellement dans un communiqué commun publié le 8 février. Il faut dire que toutes les autorités de la concurrence avaient annoncé des enquêtes sur cette acquisition record dont aucun acteur de l'industrie des semi-conducteurs ne semblait vouloir, à l'exception de Softbank et Nvidia. Retour à la case départ !

Les services Google qui seront supprimés en 2022.

Comme chaque année, Google arrête de nombreux services. 2022 sera aussi riche que 2021 :

- Déjà morts : Android Things, Cameos, Floc
- Décembre 2022 : YouTube Originals, OnHub
- Mai : G Suite (édition gratuite)
- Juin : Chrome Apps
- Mars 2023 : Google Currents

Les partenaires 2022 de

PROGRAMMEZ!

Le magazine des dévs

Niveau maître Jedi

softfluent

Scaleway

The cloud that makes sense

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

mars

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	**17** DevCon	**18** Mobilis in Mobile (Nantes)	19	20
21	22	23	**24** Cloud Sud	25	26	27
28	29	30	31			

avril

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

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** SOCRATES / Drôme	**18** SOCRATES / Drôme	**19** SOCRATES / Drôme	**20** AFUP Day / Lille	21	22
23	**24** MixIT / Lyon	**25** MixIT / Lyon	26	27	28	29
30	31					

Les événements Programmez!

Meetups Programmez!

Les dates du 1er semestre :

8 mars : Design System, REX

5 avril : Drupal & Infrastructure as Code / Christophe Villeneuve

10 mai : à venir

28 juin : à venir

Où : Devoteam 43 bd Barbès, Paris
Métros : Château Rouge (ligne 4)
A partir de 18h30

DevCon Serverless + Infrastructure as Code + Kubernes

17 mars

Où : Campus de l'école 42
96 bd Bessières, Paris
Transport : Porte de Clichy (REC C, ligne 13, ligne 14)
T3b : arrêt Honoré de Balzac
A partir de 13h30

Conférence Metavers

14 avril

Où : Campus de l'école ESGI
242 rue du Faubourg Saint-Antoine, Paris
Transport : Nation (RER A, Métro 1, 2, 6, 9)
A partir de 13h30

Informations & inscription : programmez.com

A VENIR

- AlpesCraft : 9-10 juin / Grenoble
- Le camping des speakers : 9-10 juin / Golfe du Morbihan
- DevFest Lille : 10 juin / Lille
- JFTL 2022 : 13 & 14 juin
- France API : 14 juin / Paris
- Serverless Day : 23 juin / Paris
- Hack in Paris : 27 juin – 1er juillet / Paris
- SunnyTech : 30 juin & 1er juillet / Montpellier
- JUG SummerCamp : 9 septembre / La Rochelle
- Volcamp : 13-14 octobre / Clermont Ferrand
- DevFest Nantes : 20-21 octobre / Nantes
- Codeurs en Seine : 17 novembre / Rouen

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

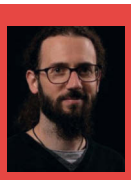
AWS Summit Paris 2022

L'AWS Summit Paris aura lieu le mardi 12 avril au Palais des Congrès de Paris, Porte Maillot. Au programme, vous pourrez retrouver des conférences thématiques, des retours d'expérience de grandes entreprises ou encore des workshops autour des technologies Cloud.
Site : https://aws.amazon.com/fr/events/summits/paris/

Nvidia GTC 2022

La GTC 2022 vous invite à quatre jours de découverte. Retrouvez-nous à cette conférence-phare pour découvrir les tendances technologiques qui transforment toute l'industrie, de la puissance phénoménale de l'IA aux techniques de collaboration virtuelle en passant par NVIDIA Omniverse et bien plus encore. Site : https://www.nvidia.com/fr-fr/gtc/

Java 18 : quoi de neuf ?



Loïc Mathieu
Consultant, formateur et speaker
https://twitter.com/loicmathieu
https://www.loicmathieu.fr

zenika

Java 18 sortira le 22 mars. Cette release ne contient pas beaucoup de JDK Enhancement Proposal (JEP), et donc pas beaucoup de nouveautés importantes. Elle arrive après la version 17 qui est la dernière version, en date, Long Term Support (LTS). Elle peut être vue comme une version de stabilisation.

JEP 400: UTF-8 by Default

Avec la JEP 400, le charset par défaut devient UTF-8 pour tous les OS et dans toutes les locales. Il est possible d'utiliser la variable système file.encoding pour configurer un charset par défaut différent, ou de la configurer à COMPAT pour revenir au fonctionnement précédent.
Plus d'informations dans la JEP-400 : https://openjdk.java.net/jeps/400.

JEP 413: Code Snippets in Java API Documentation

Ajout d'un nouveau tag JavaDoc @snippet qui peut être utilisé pour définir un fragment de code. Celui-ci est plus flexible que le tag @code existant, n'a pas besoin d'échapper les caractères spéciaux, et permet même d'inclure des fragments de code depuis un fichier externe.
Exemple de fragment défini au sein de la JavaDoc :

```
/**
 * The following code shows how to use {@code Optional.isPresent}:
 * {@snippet :
 * if (v.isPresent()) {
 *     System.out.println("v: " + v.get());
 * }
 * }
 */
```

Exemple de fragment défini en dehors de la JavaDoc :

```
/**
 * The following code shows how to use {@code Optional.isPresent}:
 * {@snippet file="ShowOptional.java" region="example"}
 */
```

Ce fragment pointe vers la section exemple du fichier ShowOptional.java :

```
public class ShowOptional {
   void show(Optional<String> v) {
      // @start region="example"
      if (v.isPresent()) {
         System.out.println("v: " + v.get());
      }
      // @end
   }
}@end
}
}
```

Il y a plusieurs options possibles de formatage et d'inclusion des fragments externes ; vos fragments deviennent alors exécutables, vous pouvez même les définir dans des tests JUnit par exemple, pour qu'ils soient testés en même temps que le reste de votre code.
Plus d'informations dans la JEP-413 : https://openjdk.java.net/jeps/413.

JEP 408 : Simple web Server

Cette JEP ajoute à la distribution OpenJDK un serveur web minimaliste qui permet de servir des fichiers statiques depuis un répertoire via un outil ligne de commande : jwebserver.
Par défaut, ce serveur web s'expose sur localhost (configurable via -b), utilise le port 80 (configurable via -p), et sert des fichiers statiques dans le répertoire courant (configurable via -d).
Chaque accès à celui-ci sera logué dans la sortie standard.

En plus d'un outil ligne de commande, vous pouvez utiliser la classe SimpleFileServer pour instancier un serveur web via son API Java.

```
var server = SimpleFileServer.createFileServer(new InetSocketAddress(8080),
      Path.of("/some/path"), OutputLevel.VERBOSE);
server.start()
```

Plus d'informations dans la JEP-408 : https://openjdk.java.net/jeps/408.

JEP 421: Deprecate Finalization for Removal

La finalisation est le mécanisme par lequel il est possible d'effectuer automatiquement des actions juste avant la destruction d'un objet. Ce mécanisme a un ensemble de problèmes, entre autres, il délaie le moment où ces actions sont effectuées au passage du garbage collector et une méthode finalize() mal codée peut faire revivre la référence à détruire.

Depuis Java 9 et la dépréciation de la méthode finalize() de la classe Object, il est déconseillé d'utiliser la finalisation et, pour libérer des ressources liées à une classe, il est conseillé d'utiliser d'autres mécanismes tel que le try-with-resource ou la nouvelle API Cleaner.

La JEP 421 va un pas plus loin dans la suppression de la finalisation en dépréciant pour suppression le mécanisme

de finalisation lui-même. Vous pouvez dès à présent tester votre code sans finalisation via l'option JVM — finalization=disabled.
Plus d'informations dans la JEP-421 : https://openjdk.java.net/jeps/421.

Des changements internes, de la performance, et de la sécurité

Chaque nouvelle version de la JDK apporte ses optimisations de performances (entre autres GC et méthodes intrisics), et de sécurité. Celle-ci ne fait pas défaut, et on peut citer entre autres des optimisations dans G1 GC (amélioration de la gestion des remember set) et Parallel GC, ainsi que dans la gestion des encodings des chaînes de caractères.

Un autre ajout notable est le support de la déduplication de String pour les garbage collectors Serial GC, Parallel GC et ZGC. La déduplication de String est un mécanisme par lequel un garbage collector va détecter des String identiques dans la mémoire heap, et dédupliquer leur tableau de byte (sans toucher aux objets String en eux-mêmes) permettant un gain d'utilisation mémoire important.

On peut aussi noter deux JEP importantes sur des mécanismes internes de la JVM :

- **JEP 418 : Internet-Address Resolution SPI** : possibilité de changer l'implémentation par défaut de résolution des adresses internet.
- **JEP 416 : Reimplement Core Reflection with Method Handles** : précédemment il y avait trois implémentations permettant de faire des appels de méthode dynamiquement : VM native method (interne), dynamic bytecode et Unsafe (utilisé par java.lang.reflect), et pour finir les method handles (java.lang.invoke). L'API de reflection (java.lang.reflect) a été réécrite pour utiliser l'API MethodHandle pour en supprimer une.

Les fonctionnalités qui restent en preview

Les fonctionnalités suivantes restent en preview (ou en incubator module).

Vector API : troisième incubation de la fonctionnalité. Il s'agit d'une nouvelle API qui permet d'exprimer des calculs de vecteur (calcul matriciel entre autres), qui seront exécutés via des instructions machines optimales en fonction de la plateforme d'exécution.
Plus d'informations dans la JEP-417 : https://openjdk.java.net/jeps/417 .

Foreign Function & Memory API : deuxième incubator pour ces deux fonctionnalités qui sont maintenant liées (l'une utilisation l'autre) au sein d'un même incubator. Foreign Memory API permet de gérer des segments mémoire (on heap ou off heap) tandis que Foreign Function permet l'interconnexion de la JVM avec du code natif (en C par exemple) de façon facile et performante. Ces deux API sont les bases du projet Panama.
Plus d'informations dans la JEP-419 : https://openjdk.java.net/jeps/419.

Pattern Matching for switch : seconde preview avec quelques améliorations. Le support du pattern matching dans les switch permet d'écrire un switch sur le type d'une variable, donc sur sa classe / son interface. En même temps, le support des null a été ajouté, on peut maintenant écrire un case null dans un switch.
Plus d'informations dans la JEP-420 : https://openjdk.java.net/jeps/420.

Divers

Divers ajouts au JDK :

- Duration.isPositive().
- Math et StrictMath ont vu l'ajout de nombreuses nouvelles méthodes : ceilDiv, ceilDivExact, ceilMod, divideExact, floorDivExact et unsignedMultiplyHigh.

Conclusion

Cette nouvelle version de Java n'apporte pas beaucoup de nouveautés, mais fait avancer le projet Panama (Vector API, Foreign Function & Memory API). On espère pour Java 19 avoir les premières JEP des projets Loom (lightweight thread) et Valhalla (value/inline/primitive classes) qui sont en cours d'écriture. Il faut quand même noter la dépréciation de la finalisation pour suppression, même si sa suppression n'est pas pour tout de suite, c'est quand même un signe fort dans cette direction.

1 an de Programmez!

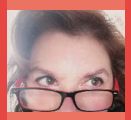
ABONNEMENT PDF : 45 €

Abonnez-vous directement sur
www.programmez.com

Who is afraid of Cybercrime? Best Practices from Cloud

RÉSUMÉ

Et si au lieu de craindre la cybercriminalité, on y faisait réellement face ? Le développeur joue son rôle et les adaptations dans le code seront importantes ou réduites. Il n'y a pas de règles. Par contre, il y a des bonnes pratiques à connaître et à réellement utiliser, notamment dans le Cloud.



Antonella Blasetti

I love to team up. With a bit of effort and pride, I managed to interact well with either young lions and IT dinosaurs. With dinosaurs it's easy, due to my long experience. The path towards lion cubs is and was through communities and humility. I like Software and Cloud Computing, which gives more power and reach to software.

I love to teach, because it is a process that can amplify knowledge, skills and cooperation.

https://www.linkedin.com/in/antonellablasetti/

https://blasetti.cloud/

Cybercrime is booming. It is important to be informed but the alarmist tone may create a state of anxiety, so rejection. This is a mistake: with a minimum of awareness, most problems could be avoided.

Let's do something unusual. Let's explore the most relevant topics in an intuitive and sometimes cheerful way. The article in short: a little awareness, attention and knowledge of the tools is enough to create the right safeguards and avoid most of the failures. Cloud Computing can help, even without any need to migrate your workloads.

These are the questions we are talking about:

- Why is there this cybercrime explosion? Should I be scared? What do I risk?
- Cops and Robbers
- What happens after your data and identity have been stolen?
- What are the threats and who can protect us?
- Does Cloud Computing help me or is it just one additional danger?
- What should I do as a programmer?
- Cloud security Tools and Best Practices.
- Cryptography. Why use it? How does it work?

Why is there this cybercrime explosion? Should I be scared? What do I risk?

Cybercrime swelled by 500% since the beginning of the pandemic. Shrewd and evil guys have increased, but there are also a lot of "good guys" who are experienced and very organized, and sometimes a little mysterious. It is a violent adventure, akin to video games.

Where there is money there is crime. Where strong emotions arise, negativity also emerges. If there are fewer bank branches, the robbers find alternatives.

In other words, crime has always been a part of life. Yet most people's lives are not centered on danger.

Thefts, identity substitutions (do you remember "the Terminator" or "Diabolik"?), blackmail, personal image damage (poor Nero!) were certainly not born with the Web!

Furthermore, swindles, spectacular thefts and any kind of violence have always fed literature and movies.

Cybercrime is no exception.Therefore, investigations can be exciting and full of magical twists.

But crime is always execrable.

We ordinary people just need to raise our level of attention.

We technicians must work and advise in a safer way, but above all we must be better organized. There is no need to write more code, but we need to collaborate within teams and technical communities, which are a great resource for us all.

In this article we will go through the basics and we will outline practical examples taken from Cloud Computing.

During this path we will find out some very easy, powerful and convenient resources to use and try. Fearless.

Cops and Robbers

Let's find out who these cybercriminals are and what they can do.

Let's start by mentioning some fun cybercrimes.

The most compelling case is Stuxnet, a virus that infected Iran's nuclear facilities and ruined their atomic bombs projects. They made a movie about it (Zero Days). Guess who the hackers were?

But Jeff Bezos also seems to have been the victim of phishing, thanks to an identity theft against Saudi Arabia's Crown Prince Mohammed bin Salman. Bezos believed he was receiving a whatsapp message from the prince, opened a movie clip and his private contents began to fly, towards rapacious hands. Mark Zuckerberg had his password cracked, too. The private data of celebrities and important people have been stolen by tampering with the systems of renowned law firms. But who are these hackers and why do they do it? Here is a list. Try to guess the good and the bad ones. It's not always that easy.

- Criminal organizations: theft, extortion, social engineering, ransomware. They also work on hire and sell easy and affordable methods to would-be scammers
- Insiders: trusted people who act maliciously or negligently
- Government intelligence and security agencies: espionage and military operations (cyber warfare or information warfare). Hence, they can be good or bad. For example, here are described cute North Korean activities
- Police organizations against crime
- Private spying - ordinary people using software to spy on other people's SMS, calls, messages, video and audio ... sometimes with devastating outcomes https://ieeexplore.ieee.org/document/8418618
- Activists, called Hacktivists; they claim to act for social change
- Vulnerability researchers who use their security expertise to discover new threats. Why? They are paid, and very well, when they are successful. There are vulnerability reward programs that give rewards when new viruses and vulnerabilities are reported and fixed.

© Caleb Woods on Unsplash

Photo by JC Gellidon on Unsplash

Photo by engin akyurt on Unsplash

What happens after your data and identity have been stolen?

They often sell them on the Dark Web so that the victim can be laughed at and robbed.
That is why it is important that data is always encrypted.
For many reasons, but also not to look like fools.
But fortunately ordinary people aren't worth all this trouble.
Even if it is an increasingly easier activity.

What are the threats and who can protect us?

Here is a list of the most popular threats, often carried by viruses:

- DDos - block website
- Brute force attacks - bots that keep on trying to get into your systems
- Identity theft and Phishing
- Insertion of Malicious Code: mainly, but not only, XSS and SQLI; the big threat for developers
- Ransomware - Steal, block and cypher your Data
- Man in the Middle - network connections in which the attacker inserts himself in the communication
- Data: breaches, exfiltration and theft

Let's reveal more about these sparkling gems:

DDos or Distributed Denial-of-Service attacks are an avalanche of hits on a website so that it soon becomes unavailable. They do not use their resources, because the attackers are hosted on servers (botnets) infected with special viruses. The server owners are completely unaware.
The victims are often large companies or institutions.

Brute force attacks are especially directed to small websites, usually with open SSH connections. They keep trying to guess the password of the root / admin accounts. If your site has minimal security (no public IP and SSH, an API or WAF gateway) it's not a big risk. But it is advisable to have a WAF that blocks them right away.

Identity theft and phishing are the door to a lot of potential harm. Pay close attention to attachments and use MFA (check your phone or press your finger) whenever possible. Take care of yourself.
And NEVER trust strangers who want to give you money. If you want to smile: This is what happens when you reply to spam email

Malicious code insertion: as just mentioned, it is a major threat to developers; so we will dive into it.

Ransomware - Steal, lock and encrypt your data. There are 3 important defences:

1. Secure identities and private address (with a strong assistant who deals with the public, for example, an API Management)
2. Regular and multiple backups
3. Encryption

Man in the Middle - the attacker inserts himself in the communication between client and server or processes. SSL / TLS is the common technique to use.

Data - breaches, exfiltration and theft. Encrypt, Encrypt, Encrypt.

On the opposite side, there are the Defenders. International organizations, security companies and ethical hackers cooperate with the giants of the Web to inform us and give us the correct tools, which everyone, from users to technicians, is advised to use. An example is MFAs - Multi-factor authentication, that many sites and especially banks impose on us. There are many non-profit or governmental organizations that study and publish descriptions and remedies for cybercrime and vulnerabilities. The greatest risk is zero-days, which are viruses and attack techniques that are active but not yet detected. All Cloud Vendors and Antivirus companies are actively working together in this fight. It's like a crime / spy series that never ends...

Does Cloud Computing help me or is it just one additional danger?

There is a common belief that Cloud Computing, being placed in a Web environment, is more dangerous than keeping everything at home. That might be true, but only if there are no targeted attacks and if all connections happen inside the data center, as was once normal.
It is not necessary to be connected to the Internet to be attacked. It may be done with just a USB stick (Stuxnet) or even a seemingly harmless USB cable (check out this nice video How You Can Be Hacked Using a USB Cable). If people connect from the outside, the system must still be defended as if it were in the Cloud.
Cloud Vendors pay the utmost attention and interest to security. For them it is vital and they experience attacks on a daily basis.
So they offer the same fundamental guarantees in a similar way.
I'm writing a series of articles on Medium comparing the various security services of different vendors.
Let's think for a moment: What is Cloud? Computing services (Virtual Machines) with a pay-as-you-go policy. OK. But not only. There are also managed and serverless services, which allow us to immediately use advanced features already installed and available, such as automatic vulnerability control and protected storing of codes and artifacts. Cloud Computing also offers easy to use processes for automated and controlled livecycle of deployments (CI/CD), which are good ways to protect our work.
Cloud Computing is designed to let us develop as little as possible. And this also applies to the security functions. Software is a liability. Maintenance and bug fixes are a boring curse for any developer.
And the IT explosion does not allow us, as in the past, to go back and work on the same things often.
So you have to ask yourself ... is this feature specific, unique to my app or something everyone needs?
In the second case it must not be developed but only integrated or simply organized. Cloud is designed to provide you with advanced functions without having to develop, install or include them in your code. Maybe with a facade or a proxy if you want to be more independent. We will talk about it soon. Not only. As for the essential services that we may need, costs are often really popular. Everything gorgeous? No problem? NO. If we are superficial and careless, it could be terrible, much worse than in the office.
Cloud services are paid and it can happen, for example, that your account is stolen and happily used to mine cryptocurrencies. With terrible economic losses. For you.

What should I do as a programmer?

As a programmer it is important to be aware of Cross-site scripting (XSS) and SQL injection (SQLI) and in general of everything that can happen when taking advantage of input data from web forms.

You have to deal with these data chunks and customer cookies. These types of problems always top the OWASP and SANS lists of common security vulnerabilities.

You can refer to this link for a detailed description and code examples. What could possibly happen?

With XSS: Acquisition and bad use of identification or session data (cookies), installation of Trojans, injection of links that redirect your users to bad places, various kinds of altered content.If you use programming languages like C ++, you need to take care to access only valid memory locations (buffer overflow).

With SQLI you risk damaging your data. Badly. If some evil guy append to a parameter something like:

```
"Or 1 = 1"
"Somedata; delete * from mytable "
```

and you happily and successfully execute these statements...Well, it might not be so nice... **Picture 1**

The solution is quite simple, in a nutshell. You have to properly escape and sanitize input data and use query parameter bindings for SQL.

SQL parameter bindings means leverage placeholders linked to typed variables in SQL statements executed from your code. There are a lot of libraries that may help.

The good news: you don't necessarily have to change your code. Maybe TONS of code.

Premise: it is a common and collective problem. It's not something a developer has to do alone, but methods and practices need to be centrally organized and shared across teams and communities. Also for javascript on the client side, even if new frameworks are currently well equipped.

Solution: API Management and Cloud WAF can take care of all of this. They scan all web requests, which can be blocked or sanitized, and their routines are continually updated to better detect and correct malicious traffic. No need for code changes and with proper logging and alerting.

Cloud may be comfortable, definitely. If you are saying to yourself .. I don't want to pay, be happy. It might be almost free. Read further.

About the code you will write in the future, keep in mind that modern methods and techniques are also the best to protect you safely:

- Immutable code
- Microservices
- Containers
- Code reviews, which help in applying best practices, are a factor of quality and optimal organization within projects.

We will talk about it again in the section dedicated to the Cloud.

Cloud security Tools and Best Practices.

Let's examine the most recommended and popular techniques of Cloud Computing and try to understand their advantages, especially from a security standpoint.

Microservices

Currently very fashionable, even if they have some drawbacks.

But microservices allow one really relevant consequence. They make it possible for huge, always online applications to evolve without touching the running code, which is updated quite little in a rolling way and without interruptions.

Write less code and achieve more. One of the Cloud's slogans is "code is a liability". It means that you'd better develop as little as possible, because maintenance will haunt you like an angry ghost. This is their strongest reason for existing. They also help to separate and organize common functions, such as access to data and services in an efficient and non-redeeming way, and therefore manage security in a more controlled and managed way.

Did you know that Cloud Computing was born for this very reason?

Amazon could no longer manage its site, so they built a new architecture and then decided to market this experience.

When a microservices architecture is hybrid it often uses proxies, which are fantastic objects, as you will see.

Containers

These nice boxes offer many advantages, but in particular they allow you to have an automated development / deployment cycle that often involves checking for vulnerabilities and fraudulent changes within them. The containers, when put in repositories, are also signed in order to avoid counterfeiting (see the cryptography section).

Facade Services

Useful to avoid exposing your server directly to the web. Your domain name will point to a more secure service and you may avoid direct access from the Internet by giving them only private IP addresses. Examples of these services are, for example, Load Balancer and API Management.

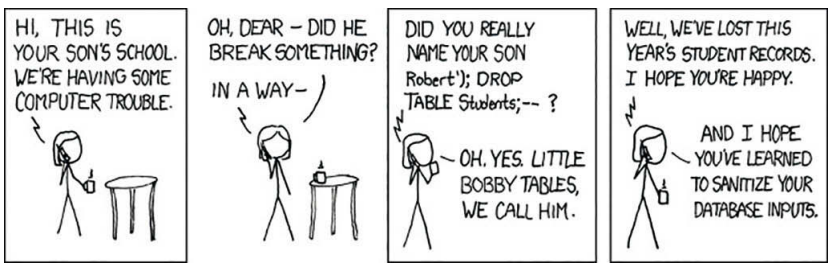
It's like having a strong assistant who takes care of all your duties and passes you only the good ones, throwing punches at those who dare disrespect you. What all of us women would love to have...

Proxy

What are proxies? To understand them we need to talk a little about networks. Networking is about connecting and communicating between nodes (servers or services).

Some site groups are protected. It means they have a fence around them. How do you enter? Through protected doors.

Picture 1



Source: https://imgs.xkcd.com/comics/exploits_of_a_mom.png

Think of border control at an airport.
You have to pass the police check. It is a gateway. He checks your passport and then you go.
If, on the other hand, they think you are suspicious, they take you to a small room and various guys intervene, rummage through your suitcases and put you in your underwear.
These nice people are proxies. They perform more tasks.
The advantage is that they have no close ties to the function to which they pass control. They help you meet new security and reliability requirements without app changes and minor changes to distributed system configurations.
For your info, the main problem with proxies is that they can become a single point of failure, and this needs to be managed.
Now let's see the most useful Cloud services for us and why and how to best exploit them.
We will look into "Why Cloud Computing" right away. Let's imagine we want to check all the traffic to one of our apps; we find the right software and install it in a VM with its nice Listener. Let's imagine what may happen:

- We get a fierce attack and our VM dies, killing our app, too
- Security checks need to be often updated and therefore also our software. Do we need to stop everything? Scared of a bad outcome?
- Something strange is happening, but it could be a false alarm. How to handle it quickly?

None of this is unfeasible, but think about the amount of work you have to put into it. In the Cloud everything is ready to use and, at least for basic services, affordable.
Let's see the solutions that are implemented with Cloud (but also in the home and hybrid) with a very intuitive explanation of what you can do.
Log and Monitoring
API Management and Service Management
WAF - Web Application Firewalls
Online Detection Systems
Historical Investigation and dangerous resources detection
Configuration Analysis

Log and Monitoring
Logging is automatic, configurable, extensible, keeps track of everything and can unify the structure of different kinds of logs. An open source product called Fluentd is often used.
When something happens, they are really helpful in realizing what happened.

API Management and Service Management
Amazing tools, different but with many analogies. Let's see immediately how they differ:

API Management Systems are linked to applications (the famous REST APIs) and act as an interface between client and server app.
They are the ones who expose a domain name and then call programs that may be located in different places, even on-premises. They manage SSL, authentication, throttle annoying bots, manage security, and mitigate attacks.
Service Management (ever heard of ISTIO?) Are used in organizations where thousands of distributed microservices are used, usually with Kubernetes, for different applications.
The goal is not so much the management of a backend but being able to control and secure such complex systems.
They are also referred to as Service Mesh. Yes, like very tightly meshed nets.
They have many things in common. First of all, proxies, programs that carry out security checks and operations and record what happens (logs and telemetry).
Which are managed independently from your app, so they can be updated easily.
So they provide the ability to read and modify requests and responses, with ready-made proxies that detect abnormal behavior and block them.
They can also call, when needed and without disturbing your app, fixing routines.
Your apps keep on living quietly just doing what they were written for. And nothing else.
These automations are common all over Cloud Computing and it is all tracked in the logs… in order to monitor what happened. API Management Services are very convenient and efficient. And they can also be used for apps that are not in the Cloud. Security gurus will turn up their noses, but getting traffic from Google or AWS is different from being exposed to freezing cold.

WAF - Web Application Firewalls
WAF are amazing! In Google they are called Cloud Armor and in AWS they are simply WAF.
They also catch all the requests that may be blocked if they come from unwanted places; in addition, they have a lot of suggested OWASP checks inside.
Here too, they protect against unwanted BOTs.
But how are they different from API Management?
First of all they work together with the API, because the WAFs are configured together with a Load Balancer (the service that distributes requests to multiple VMs or containers, therefore network stuff) or with an API Management.
So we can put them together and strengthen our defenses.
They only work inbound, and their job is mainly that of bouncers that don't care about or alter your app responses.
They are guards. But their logic is very clever (they use ML) and continuously updated.
Last but not least ... popular prices!

Continuous Monitoring - Intelligent threat detection
They continuously check your Cloud resources looking for strange and dangerous behaviours.
The difference is that here we are not limited to dangerous requests but we analyze all the operations in the virtual Data Center. If there is too much activity, if too many systems are created, if there is any atypical processing. At the heart of your systems.
A simple example: if suddenly coin mining activities are started in a VM that usually deals with accounting, you have the opportunity to notice and block them quickly.
The products are different: AWS has GuardDuty and Google has Anomaly Detection and Container Threat Detection.
Costs are higher.

Historical Investigation and dangerous resources detection

But what if something happened in the past and we didn't notice? And maybe this monster is there in the shadows waiting for the right moment to start their mission of destruction!

No fear. These systems, when new threats are discovered, go to scan into the past if there is loathsome lurking.

How do they do it? With the logs, of course.

Technically they are very sophisticated systems that process large amounts of data with ML techniques. And they cost. But of course they are for big enterprises.

Google Chronicle and AWS Detective are our examples.

There are many other services, especially those for identification and authorization, but we looked into the most significant and intriguing for the "criminal" theme.

Cryptography. Why use it? How does it work?

Encryption is not aimed at avoiding attacks, but at information protection.

They don't have to get to your private or your organization's affairs. Definitely.

But, as already mentioned, Encryption may save your reputation and career. Not bad.

It is essential to understand encryption keys. These keys are exactly like the combinations of safes or the ones you put in a keyhole.

And, likewise, they are the weak link in the chain. If the key goes into the wrong hands, it is a big trouble.

In computer science there are 2 problems: the conservation of these keys and their transmission.

These problems, especially the second, made many famous mathematicians go through sleepless nights.

Take care! Encryption keys are different from passwords, or secrets. They are only used to encrypt numbers or texts.

Let's start from basics, or rather, from what we did as kids. In order to keep our secrets, we used to speak by replacing words or letters. For example, let's say we shift 2 alphabetic positions for each letter: abc becomes cef - home becomes jqog. So you have a method (simple shift) and a key (2).

In spy stories our heroes or villains had to discover both the methods AND the keys. But the web is public, so methods are shared; therefore the keys are vital.

About conservation, the keys are kept in electronic safes (vault) which sometimes are in specific hardware (HSM) and super-certified.

And they can't move from there.

But then what good are they if not used?

Is a glittering tiara needless if it doesn't light up a beautiful princess?

All carefully planned. These systems provide methods for secure transmission of data to be encrypted without moving the keys from their realm. Like queen bees.

Instead, hold on because the transmission part is awesome.

The turning point came with asymmetric keys: one public (weaker) and one private (stronger);

They are not that different, they are just siblings, or at best cousins.

Making it simple: the public one is a large number, the private is a super-large number composed of several prime numbers which, when multiplied, make the number of the public key. So the private one can derive the public one (a multiplication is enough) because it knows operands.

With the public key alone, too much time and effort is needed to figure out what the right combination of factors really is.

At this point the encryption algorithm takes over, it is designed in such a way as to be able to encrypt a text with the "weak" key but cannot decrypt it because it would need the operands. So the message is readable ONLY with the public key. The two keys are therefore linked. And math can create this magic. Another sleight of hand is to create a "simple" key that can both encrypt and decrypt without ever transmitting it over the Internet, keeping it safe.

How do they do it?

They start from a random number that will be processed by both the client and the server but with different keys, those owned by them. The 2 different results are exchanged and reprocessed with the other key.

Thanks to the particular mathematical procedure, the resulting number will be the same on both sides, without ever having been transmitted on the Net.

Only the result of the formula was exchanged which, if guessed, did not allow to go back to the secret numbers.

This "simple" key is the foundation of TLS / SSL, which uses asymmetric cryptography to define a common key that is never shared publicly. This new key will be used to quickly encrypt data with symmetric methods.

The most effective and used intuitive explanation, taken from Diffie–Hellman_key_exchange, is the following which uses colors. **Picture 2**

The last important concept is represented by electronic signatures.

The mechanism is the same as encryption with public / private keys, which is also valid in reverse, that is, you can encrypt with the private key and decrypt with the public.

Once you understand this, it's all very simple.

As for handwritten signatures, the subject declares its name, in clear text, paired with the encrypted version, made with the private key (which only he has).

Whoever wants to verify the identity reads the name, decrypts the encrypted part with public key, obtaining the same name. Voila. Only the signer, who has the private key, can create its own encrypted name.

All this to avoid that, during all the travels of data packets, someone pops up, gets in the way and steals identities… and values.

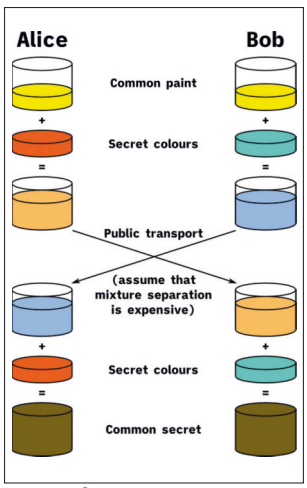
Conclusion

We have covered in a very simple way the main topics and solutions regarding CyberSecurity.

There is much more, explained in a more rigorous (and difficult) way. But, as in life, it is often enough just to know the essentials and to use common sense.

Stay safe from all kinds of viruses. Be happy.

Photo by Alex Motoc on Unsplash



Picture 2

Marylise Tauzia

Marylise fait partie de l'équipe de développeurs de Square depuis plus de 5 ans. Basée à San Francisco, elle manage les activités globales d'évangélisme pour la plateforme de Square et les relations avec les développeurs externes. Elle a reçu un diplôme d'ingénieur de l'Université de Technologie de Troyes avant de déménager à San Francisco. Elle est passionnée par le domaine du FinTech et son travail avec les développeurs du monde entier pour collaborer avec eux et aider les commerçants à innover et gérer leur business.

Paiements multicanaux avec React & Square

Depuis quelques années, l'évolution rapide des moyens de paiement force les commerçants à moderniser leurs systèmes et à offrir de multiples options pour prendre les commandes et les paiements associés. Et la pandémie n'a fait qu'accélérer cette tendance.

Que ce soit les commerces de proximité, la vente au détail, les restaurants… tous les commerçants ont besoin d'offrir des options de vente en ligne ou sur application mobile pour survivre et à l'inverse, nous observons aussi que les commerçants qui ont démarré en ligne, veulent ouvrir un magasin pour se développer et fidéliser leurs clients. Avoir la capacité de prendre une commande de n'importe où ainsi que son paiement est plus important que jamais. Square offre une plateforme flexible qui permet aux développeurs de mettre en place des solutions adaptées. Dans cet article, nous allons détailler comment mettre en place une opération complexe : "la gestion de commandes passées à l'avance et à distance". Cette opération est aussi connue sous l'acronyme anglais BOPIS "Buy Online Pickup In Store" - "Achat en ligne, retrait en magasin" pour les commerces de détail et "Order Ahead - vente à emporter" pour les restaurants.

Paiements en ligne et sur application mobile avec React

Pour mettre en place une solution de paiement multicanale, j'ai décidé d'utiliser React et React Native, car ces langages de programmation vont me permettre de développer ma solution en ligne avec le React Square Web Payment SDK et sur application mobile avec le React Native In-app Payments (IAP) SDK. Le React Square Web Payment SDK a été créé par Seed, et officiellement supporté par Square. Il est utilisé pour prendre la commande et nous permet d'afficher plusieurs options de paiement qui capturent de manière sécurisée les détails du client. Avant de démontrer comment prendre le paiement, je vais d'abord décrire les étapes qu'un client qui veut acheter quelque chose sur un site va suivre. Tout d'abord, le client arrive sur la page d'accueil du site qui affiche les produits à vendre et qui sont stockés dans le catalogue. Le client peut cliquer sur un objet pour l'acheter. Pour compléter l'achat, il doit entrer ses informations personnelles et de livraison ainsi que son numéro de carte de crédit. Une fois les informations validées, une commande va être enregistrée par Square et apparaîtra dans le gestionnaire de commande sur le point de vente Square du commerçant. Le processus d'exécution de la commande démarre à ce moment jusqu'à ce que le client vienne la chercher.

Regardons d'abord comment retrouver le catalogue :

```javascript
export async function getProducts() {
  const { catalogApi } = client;
  try {
    const { result: { objects, relatedObjects }} = await catalogApi.searchCatalogObjects({
      objectTypes: [
        'ITEM'
      ],
      includeRelatedObjects: true
    });

    const images = relatedObjects.filter(object => object.type === 'IMAGE');
    const items = objects.filter(object => object.type === 'ITEM');

    const products = items.map(object => {
      const {
        id,
        imageId,
        itemData: {
          name,
          description,
          variations,
          categoryIds,
        }
      } = object;
      const imageUrl = images.find(image => image.id === imageId).imageData?.url;

      return {
        id,
        name,
        description,
        imageUrl,
        variations,
        categoryIds
      }
    })
    return JSON.parse(JSON.stringify(products));
  } catch (e) {
    console.error(e);
    throw e;
  }
}
```

Dans le code ci-dessus, j'utilise l'endpoint SearchCatalog de l'API Catalog et je fusionne les données images avec les données produits de qui va permettre au frontend de tout afficher pour chaque produit.

Maintenant, je vais utiliser le composant React pour afficher les données dans l'application Web ou l'application mobile. J'utilise la même fonction pour les deux (avec quelques changements mineurs pour que ça marche avec React Native).

Voilà les composants ProductList :

```javascript
import { formatCurrency } from "../utils/formatting";
import Link from 'next/link'
```

```
export default function ProductList({ products }) {
 return (products.length !== 0) ? (
  <div className="bg-white">
   <div className="max-w-2xl mx-auto py-16 px-4 sm:py-24 sm:px-6 lg:max-w-7xl
 lg:px-8">
    <h2 className="sr-only">Products</h2>

    <div className="grid grid-cols-1 gap-y-10 sm:grid-cols-2 gap-x-6 lg:grid-cols-3 xl:
grid-cols-4 xl:gap-x-8">
     {products.map((product) => (
      <div key={product.variations[0].id} className="group">
       <Link href={product.id} className="group">
       <div className="w-full aspect-w-1 aspect-h-1 bg-gray-200 rounded-lg over
flow-hidden xl:aspect-w-7 xl:aspect-h-8">
          <img
           src={product.imageUrl}
           alt={product.description}
           className="w-full h-full object-center object-cover group-hover:opacity-75"
          />
         </div>
        </Link>
        <h3 className="mt-4 text-sm text-gray-700">{product.name}</h3>
        <p className="mt-1 text-lg font-medium text-gray-900">{formatCurrency
(product.variations[0].itemVariationData.priceMoney.amount)}</p>
      </div>
     ))}
    </div>
   </div>
  </div>
 ) : <span>Loading</span>
}
```

```javascript
import React, {useEffect, useState} from 'react';
import {Image, Text, TouchableOpacity, View} from 'react-native';
import tailwind from 'tailwind-rn';
import {formatCurrency} from '../../utils/utils';

export default function ProductList({navigation, route}) {
 const [products, setProducts] = useState([]);

 useEffect(() => {
  fetchProducts();
 }, []);

 const fetchProducts = async () => {
  const res = await fetch('http://localhost:3000/api/square/catalog');
  const catalogProducts = await res.json();
  console.log(catalogProducts);
  setProducts(catalogProducts);
 };

 return products.length !== 0 ? (
  <View style={tailwind('bg-white')}>
   <View style={tailwind('max-w-2xl py-16 px-4')}>
    <View style={tailwind('flex')}>
     {products.map(product => {
      return (
       <View key={product.id} style={tailwind('')}>
        <TouchableOpacity
         style={tailwind('w-full rounded-lg overflow-hidden')}
         onPress={() => {
          navigation.navigate('Checkout', {
           product,
          });
         }}>
         <Image
          source={{
           uri: product.imageUrl,
          }}
          alt={product.description}
          style={tailwind('w-full h-64')}
         />
        </TouchableOpacity>
        <Text style={tailwind('mt-4 text-sm text-gray-700')}>
         {product.name}
        </Text>
        <Text
         style={tailwind('mt-1 text-lg font-medium text-gray-900')}>
         {formatCurrency(
          product.variations[0].itemVariationData.priceMoney.amount,
         )}
        </Text>
       </View>
      );
     })}
    </View>
   </View>
  </View>
 ) : (
  <Text>Loading</Text>
 );
}
```

Il est important de noter une petite différence entre notre client Web et notre client mobile : Nous allons chercher les informations du catalogue de façon dynamique sur le client mobile alors que nous le faisons au moment du "build" pour le client Web. Cela est dû au fait que les clients mobiles sont plus lents à sortir des mises à jour et nous pouvons générer des éléments statiques plus facilement sur le Web.
Une fois que le client a cliqué sur le produit qu'il désire acheter, nous passons à la page de paiement. C'est sur cette page que nous avons besoin d'utiliser le React Square Web Payments SDK et le React Native Plug-in for In-App Payments (IAP) SDK. Nous pouvons configurer l'interface de paiement pour prendre les cartes bancaires ou cartes de crédit, Apple Pay, Google Pay ou des bons cadeaux. Regardons tout d'abord comment ajouter l'option carte bancaire ou carte de crédit. Voilà un exemple d'utilisation du In-App Payment SDK intégré dans un composant appelé 'SquarePay" :
Code complet sur programmez.com & github

Que ce soit le Web Payment SDK ou le In-App Payment SDK, tous deux capturent d'abord les informations de la carte du client et vous donnent un "callback" qui renvoie un "token" utilisable une seule fois. Vous pouvez utiliser ce "token" avec l'API Square pour soit charger la carte directement ou bien

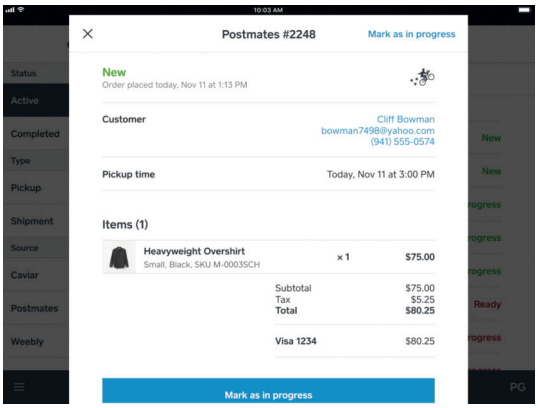


Figure 1

juste stocker les informations dans le fichier du client pour l'utiliser plus tard.

Le code ci-dessus utilise tout d'abord la méthode **StartCardFlow()** qui débute le processus de paiement. J'utilise ensuite les callbacks **onCardNonceRequestSuccess()**, **onCardEntryCancel()**, and **onCardEntryComplete()** pour valider la bonne réception du "token" et traiter le paiement avec l'API, pour annuler une entrée ou valider que le paiement a été complété avec succès.

Le code est en React Native pour mobile, mais la façon de gérer les cartes de paiement avec le Web Payment SDK est similaire.

Code complet sur programmez.com & github

Dans ce code, je montre à quel point il est simple d'ajouter une option de paiement par Apple Pay ou Google Pay en utilisant le React Square Web Payment SDK. Il suffit juste d'ajouter la fonction createPaymentRequest et inclure les composants ApplePay et GooglePay. (Il est aussi très simple d'ajouter d'autres méthodes de paiement avec quelques conditions de plus).

Dans une application en production, il sera important de mettre à disposition les formes de paiement telles que Apple Pay et Google Pay, car elles sont de plus en plus utilisées ainsi que s'assurer que SCA (Strong Customer Authentication) est en place pour vérifier l'identité de l'acheteur. Il est aussi possible de modifier le design du formulaire de paiement pour qu'il s'accorde bien avec le design de votre application ou de votre site.

Pour finaliser l'application, il faut envoyer les détails du paiement à notre "backend" pour créer la commande, la relier au client ainsi que relier le paiement à la commande elle-même. Avec Square, il est possible de gérer à la fois le paiement Web ainsi que le paiement mobile en utilisant le même "endpoint". Regardons en details :

Code complet sur programmez.com & github

Maintenant, pour le développeur, tout semble fini ! Succès ! Cependant, pour le commerçant, c'est là que tout commence ! La commande vient d'être créée et est apparue dans le gestionnaire de commandes Square (Square Order Manager) pour qu'il puisse la visualiser et la traiter. **Figure 1**

Quand le commerçant met à jour le statut de la commande, vous pouvez utiliser des "webhooks" pour écouter les mises à jour et créer des actions associées telles qu'envoyer un SMS ou une notification à l'acheteur pour lui laisser savoir quand la commande est prête ou s'il y a du retard.

Ce que j'aime le plus dans tout ça, c'est qu'en tant que développeur, je n'ai pas a construire tout ce mécanisme de traitement de commande et d'envoi de mises à jour à l'acheteur. Tout est géré par le point de vente ou dans le tableau de bord (interface Web) de Square. Cela permet au commerçant d'avoir la meilleure expérience possible et le développeur peut se concentrer sur la mise en place de la prise de commande sur des canaux multiples.

Paiement en personne avec React

Je viens de détailler comment implémenter un formulaire sécurisé de paiement en ligne ou sur application mobile avec React, regardons maintenant comment implémenter une application personnalisée pour permettre à un commerçant de prendre un paiement en personne avec Square Terminal. En utilisant l'API Terminal, on peut utiliser Square Terminal et le connecter à une application de point de vente développée en React. Tout d'abord, je vais connecter le Terminal à l'application React.

```javascript
try {
const response = await client.devices.create_device_code(body: {
 idempotency_key: '{{CHAÎNE_ALÉATOIRE}}',
 device_code: {
   name: 'Comptoir 1',
   product_type: 'TERMINAL_API',
   location_id: 'B5E4484SHHNYH',
 }
});

console.log(response.result);
} catch (error) {
console.log(error);
}
```

Et maintenant, je vais envoyer une transaction de 1€ au Terminal connecté.

Code complet sur programmez.com & github

Et voilà ! Vous avez sûrement compris qu'on ne se limite pas à React dans ce cas. Vous pouvez simplement créer un "endpoint HTTP" pour faire un appel depuis n'importe quel terminal de paiement connecté au Web pour déclencher une requête. Donc, que vous travaillez avec une application Web, mobile ou autre, vous pouvez démarrer le processus de paiement sur le Square Terminal depuis n'importe où ! Nous avons déjà vu des développeurs créer diverses applications telles que les [kiosques pour passer des commandes] (https://devpost.com/software/zaastry-with-paymenteye), [Des boutons de commande "1 touch"] (https://devpost.com/software/flash-order-one-button-ordering) et aussi des [paiements contrôlés avec la voix] (https://devpost.com/software/voice-order).

En conclusion

J'espère que ce guide vous a donné une idée de ce dont vous avez besoin pour démarrer un développement avec les API et SDK de Square pour prendre des paiements en ligne, dans une application mobile ou en personne. Vous pouvez ainsi développer des expériences qui connectent plus de clients aux commerçants. Cependant, j'ai aussi montré que nos API de paiement sont seulement une partie des API de la plateforme. Si vous voulez en savoir plus, n'hésitez pas à visiter notre espace développeur sur developer.squareup.com/fr/fr

Ruby a 10 ans !

Valériane

Developer Evangelist chez Twilio - une plateforme de communication via API - ancienne développeuse web freelance ayant toujours travaillé avec Ruby entre autres, pour vous servir.

twilio

Depuis plusieurs années, Ruby se fait plus discret mais le langage continue à évoluer et son écosystème à s'étendre, notamment avec Rails qui reste le framework le plus connu. Si le langage n'a jamais connu la notoriété d'un C# ou d'un Python et pourtant, Ruby a poursuivi sa route à son rythme. Valériane nous propose une plongée dans le langage. La rédaction.

Ruby fête ses 10 ans cette année, l'occasion de vous parler de mon langage de programmation préféré parmi tous ceux utilisés aujourd'hui.
Pour célébrer l'anniversaire du langage, je vous propose de me suivre pour le grand tour. Qu'est-ce que c'est ? Pourquoi c'est bien ? Est-ce que ça vaut le coup de l'apprendre aujourd'hui ? Gagne-t-on sa vie en écrivant du Ruby ? Vous aurez, dans cet article, la réponse à toutes ces questions et bien plus.

Un langage par et pour les devs

```ruby
print "Hello World\n" # le programme Hello World en Ruby
```

Ruby est un langage de programmation créé pour le bonheur des devs au Japon par Yukihiro "Matz" Matsumoto.
Matz programmait depuis un moment avec différents langages (Ada, Eiffel, Lisp, Perl, Python, Smalltalk) et a décidé de créer un nouveau langage de programmation qui combinerait toutes ses features préférées de chacun de ces langages. La première version de Ruby fut donc développée entre 1993 et 1995 et était dès le départ un langage de programmation de haut niveau interprété, orienté objet, multi-paradigme et designé pour être flexible et utilisable sur la plupart des plateformes (BeOS, DOS, Linux, macOS, OS/2, UNIX, Windows...).
Matz a utilisé son expérience de programmeur lors de la création de son langage, ce qui fait de l'expérience de développement l'un des éléments principaux de la philosophie de Ruby et le confort de développement l'un de ses points forts majeurs. D'un point de vue technique, si on doit retenir une seule chose, c'est qu'en Ruby, *absolument* tout est un objet.
Depuis 1995, de nombreuses versions sont publiées à un rythme assez soutenu depuis le début pour désormais sortir une version majeure tous les ans à la période de Noël et ce depuis 2015.
Il fut standardisé en 2012 par l'Organisation Internationale de Standardisation sous la référence ISO/IEC 30170:2012.

Les débuts du succès

Étant un langage créé pour le confort des devs, Ruby les as attirés depuis le départ, formant petit à petit une communauté forte autour de lui au Japon jusqu'à devenir l'un des 10 langages préférés des devs dans le monde et le langage de l'année en 2006 selon TIOBE.
Son succès n'est pas étonnant au vu de ses principes de base :

- **Principe de concision** : Le but des devs est d'avoir une petite codebase à maintenir et faire évoluer constituée de petits morceaux de code fonctionnels et lisibles. En Ruby il est possible d'écrire beaucoup d'instructions récurrentes en une seule ligne et de créer facilement ses propres alias pour tout ce que l'on voudrait raccourcir.

```ruby
books.each { |book| print book.author } # un exemple de parcours de liste en une ligne en Ruby
```

- **Principe de cohérence** : aussi appelé le principe de la moindre surprise. C'est le fait que si vous connaissez un ou plusieurs langages de programmation, le fonctionnement de Ruby vous semblera familier.
- **Principe de flexibilité** : Ruby en tant que langage de programmation ne se mettra pas entre vous et la réalisation de votre idée. Il y a un tas de façons différentes de parvenir à un même résultat et Ruby vous laisse choisir celle que vous préférez. Vous pouvez aller jusqu'à changer les commandes intégrées au langage pour vous donner une idée de l'étendue des possibilités offertes.

Avec tous ces attributs en poche, le succès de Ruby n'était qu'une question de temps et c'est 2004, l'année qui marque un tournant majeur dans l'histoire du langage. 2004 c'est

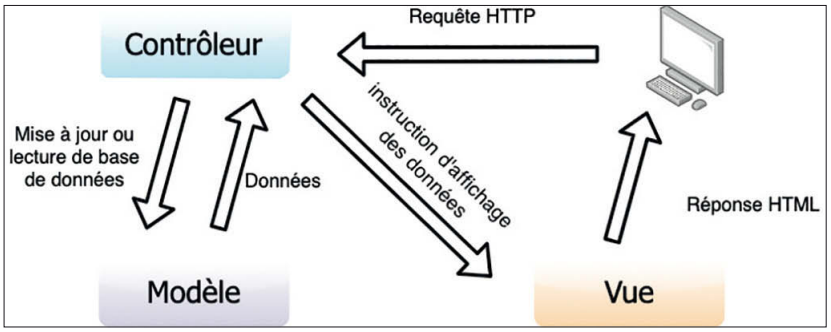


Figure 1

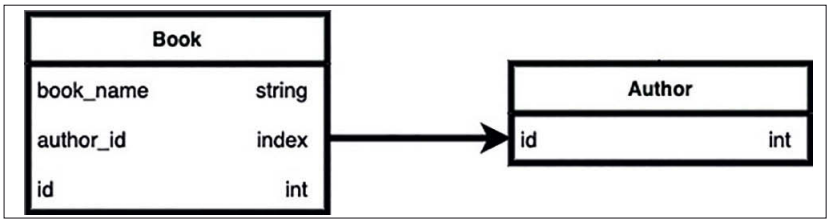


Figure 2

bien sûr l'année de la première version anglaise de la documentation, mais c'est surtout l'année où le plus connu des frameworks Ruby a été publié: Ruby On Rails !

À l'époque, aux USA, dans une société nommée Basecamp, un développeur code un énorme projet éponyme. Ce dev, plus connu désormais sous le nom de DHH, savait que le projet était très complexe et il décida donc de créer son propre framework pour mener à bien son projet.

Vous vous en doutez, il utilisa Ruby pour créer son framework et commença à l'appeler Ruby On Rails. Le succès est massif et immédiat et nous en parlerons dans les chapitres suivants, mais l'histoire aurait pu s'arrêter à ce moment-là. Ce framework aurait pu devenir un outil interne à Basecamp utilisé seulement par leurs équipes mais DHH décida de le publier sur GitHub sous licence MIT, le rendant ainsi accessible partout, pour tous.

Un langage pratique à utiliser, une communauté grandissante et un framework robuste ? Il n'en fallait pas plus à Ruby pour exploser et se faire une place sur la scène des langages reconnus et aimés mondialement.

Et même si Ruby est utilisé aujourd'hui par les plus grands, les startups furent les premières à montrer leur amour à Ruby On Rails.

Ruby & Startups

En 2006, aux USA, après la publication par BaseCamp de Ruby on Rails, le framework et le langage ont littéralement explosé. Les startups de la Silicon Valley ont fait confiance à Rails pour lancer leur produit en faisant ainsi une référence dans l'écosystème ayant les yeux rivés sur San Francisco. Plusieurs startups notables ont vu le jour et depuis, un certain nombre de licornes ont eu Ruby et Rails dans leur stack initiale.

Parmi celles dont vous avez forcément entendu parler un jour, on peut citer GitHub, Twitter, AirBnB, Twitch, Etsy, Kickstarter, Soundcloud… Bref la liste est longue et c'est indéniable que le succès de startups qui ont désormais cette envergure a largement contribué à faire parler du framework et du langage sur lequel il s'appuie.

Et même s'il y a eu des migrations totales ou partielles de la stack de certains des noms cités précédemment, aujourd'hui plusieurs l'utilisent encore et de grands noms tels que Doctolib et Shopify se sont ajoutés à la liste des convaincus de la techno.

Mais pourquoi se sont-elles tournées vers Ruby et Rails ? Voyons ensemble les points qui font la différence pour ces écosystèmes :

• Le **développement** avec Rails est **très rapide**. **Figure 1**

Ce framework est un MVC fortement orienté par un principe de "Convention supérieure à la Configuration", ce qu'il fait qu'il embarque un tas de commandes CLI de génération de code. Ainsi on peut créer un fichier contenant toutes les infos de base pour un modèle, un contrôleur, et même tout ça et plus à la fois. La commande

```bash
bin/rails generate scaffold Book book_name:string author_id:integer
:index
```

va non seulement créer un fichier pour un modèle `Book` contenant les champs `book_name` qui sera une string contenant le nom du livre et `author_id` qui sera un int mais aussi un index en base de données vers un modèle `Author` que nous pourrions créer. **Figure 2**

En plus du fichier du modèle, cela crée aussi un contrôleur `BooksController`, des vues pour chaque action CRUD liée à ce contrôleur dans un répertoire `views/books/`, ajoute la déclaration des routes HTTP dans le fichier dédié dans Rails `config/routes.rb` et crée le fichier de migration nécessaire à la création de la table Books en base de données et de ses colonnes `book_name`, `author_id` et de l'indexation de `author_id`, mais aussi les fichiers contenants les tests basiques pour ces nouvelles fonctionnalités.

C'est là juste une des nombreuses options embarquées pour accélérer le développement avec Ruby On Rails, ce qui le rend parfait pour la création d'un MVP (produit viable minimum) et donc attrayant pour l'écosystème startups.

• **Ruby on Rails est scalable**.

Pour vous donner une idée, Shopify traite plus de 4 millions de requêtes par seconde sans broncher. Il est parfait pour sortir une application navigable sur localhost en littéralement 5 minutes qui fait déjà des choses d'un point de vue backend, mais il peut vous suivre facilement dans vos aventures d'élargissement de votre base utilisateurs ou data.

Il est possible avec Rails de se connecter à plusieurs bases de données.

ActiveRecord, l'ORM de Rails, embarque des fonctions de caching.

Du fait de son architecture, Rails est aussi très clair sur où mettre la business logic (Gros modèle, petit contrôleur), comment ne pas se retrouver avec du code dupliqué (les helpers sont vos amis) et où gérer le frontend (uniquement dans le fichier vue ou connectez un framework de front et gérez tout ça dans un folder dédié). La communauté promeut aussi de nombreuses règles de style, et embarque son linter et formateur RuboCop, ce qui rend le code Ruby consistant malgré sa permissivité, et ce qui en fait un langage dont le code est

maintenable dans le temps. **Figure 3**
Bien sûr le travail de devops sera toujours nécessaire à une mise à l'échelle réussie mais des options sont présentes du côté du développement.

• **Ruby on Rails est sécure**.
De base, le framework fournit un système de sessions protégées par des cookies cryptés, tous deux configurables. Rails protège aussi des attaques de type Cross Site Request Forgery (XSRF en français) en intégrant un flag `config.action_controller.default_protect_from_forgery` réglé sur `true`, ce qui va automatiquement ajouter un token pour tous les formulaires et requêtes AJAX de l'application. Si le token est manquant ou non semblable à ce qui est attendu, une exception sera levée.
Pour autoriser les paramètres d'une requête entrante on utilisera le DSL de Rails. Pour autoriser dans un controlleur les paramètres liés au modèle Book que j'ai créé précédemment on écrira la déclaration suivante dans le controlleur:

```ruby
def book_params
 params.require(:book).permit(:book_title, :author_id)
end
```

Il serait possible, si je voulais autoriser la création ou modification sur des champs liés au modèle `Author`, d'ajouter des attributs imbriqués ou nested attributes, ce qui ressemblerait à:

```
params.require(:book).permit(:book_title, :author_id, author_attributes:
 [ :id, :name, :date_of_birth])
```

Je pourrai désormais utiliser ces `book_params` pour manipuler la donnée et l'entrer en base de données de manière plus sécurisée.
On peut aussi noter que de nombreux bootcamps de code adressés à un **public entrepreneurial,** comme Le Wagon ou The Hacking Project pour des exemples français, ont vu le jour et ont contribué à la démocratisation du langage dans l'écosystème en France et dans le monde.
Comme l'amour des startups pour Ruby nous le prouve, sa flexibilité et sa fiabilité en ont fait un langage majeur dans le paysage des langages de développement.

Tout pour réussir

La conquête fulgurante des start ups ne se serait pas faite sans des propositions robustes de la part du langage en plus de ce dont nous avons déjà parlé jusqu'ici. Intéressons-nous donc maintenant à pourquoi Ruby est l'un de mes langages préférés pour coder.
Je vous ai parlé jusqu'ici en filigrane de l'une des forces majeures de Ruby qu'il est temps de détailler maintenant. Sa communauté.
Ruby est un langage open source, gratuit, libre d'utilisation et de modification. Il existe depuis un moment et à su fédérer une communauté forte qui a contribué à améliorer le langage au fil des ans mais surtout à contribué à la richesse de l'écosystème Ruby, ses Gems.

Les gems sont des librairies externes facilement plugables à votre runtime Ruby créées par la communauté. Elles sont intégrées à la distribution standard de Ruby depuis la version 1.9 grâce à l'intégration de RubyGems, un outil de téléchargement, d'installation et de publication de ces gems. Au-dessus de ça est venu se plugger Bundler, un outil qui a introduit le fameux Gemfile dans les projets Ruby, et qui est lui-même packagé dans une gem.

Pour l'installer il faut utiliser la commande :
```bash
gem install bundler
```
Puis il suffit de créer un fichier nommé Gemfile à la racine de votre projet et de le remplir avec une source, et les Gems que vous voulez dans votre projet.
Par exemple pour intégrer la gem twilio-ruby à un projet et l'utiliser il faudrait dans un premier temps créer le Gemfile suivant :

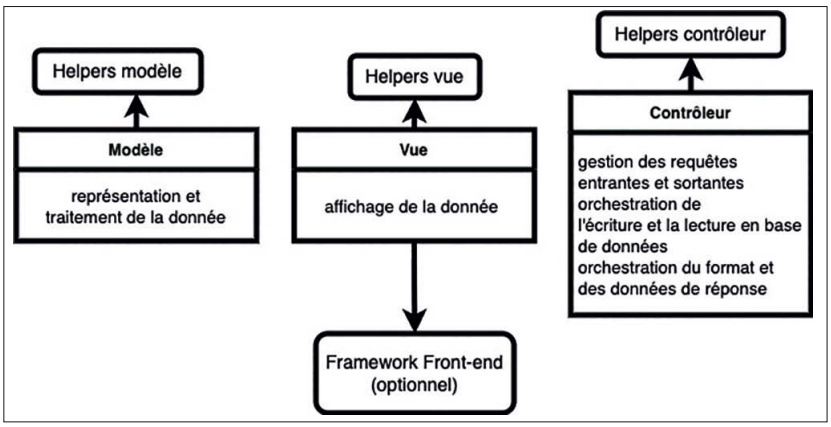
```ruby
source 'https://rubygems.org'

gem 'twilio-ruby', '~> 5.59.0' # nom de la gem suivi de la version désirée.
```

Il est possible et même recommandé de définir la version de la gem que l'on souhaite utiliser, pour des questions de compatibilité.
Puis dans un fichier que je nommerais `twilio.rb`

```ruby
require 'twilio-ruby'

@client = Twilio::REST::Client.new(ENV["ACCOUNT_SID"], ENV["AUTH
_TOKEN"]) @client.messages.create( from: '+336xxxxxxxx', to: '+336
xxxxxxxx', body: 'Salut les lecteurs !')
```
Figure 4

Aussi simple que ça. En trois lignes de code, j'ai créé un client capable de se connecter aux APIs Twilio grâce à mes identifiants stockées dans mon environnement, et j'ai envoyé un SMS sur un vrai numéro de téléphone grâce au client.

Figure 3



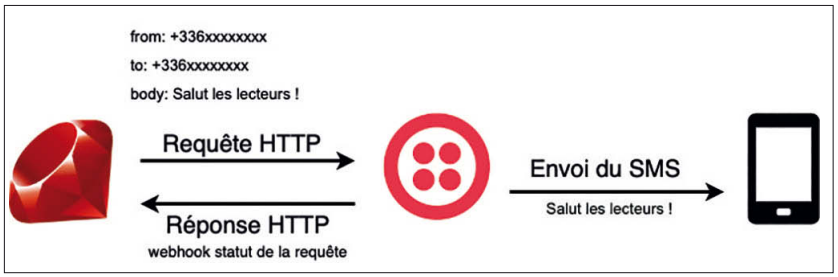


Figure 4

Pour exécuter ce code localement, je n'ai plus qu'à saisir dans mon terminal :

```bash
ruby twilio.rb
```

Il est aussi possible d'exécuter les programmes à l'aide d'un Rakefile, un fichier qui se comporte comme un Makefile en C mais avec une syntaxe Ruby.
D'ailleurs les Gems en possèdent toutes un car leur structure est la suivante :

```
gem_name/
|— bin/
|    └— gem_name
|— lib/
|    └— gem_name.rb
|— test/
|    └— est_gem_name.rb
|— README
|— Rakefile
└— gem_name.gemspec
```

Le dossier lib contient le code de la gem, le dossier test est plutôt explicite,le README aussi, le fichier `gemspec` les infos de la gem et celles de son auteur.
Quand au Rakefile, c'est l'équivalent d'un Makefile en C mais écrit en Ruby car si vous ne le saviez pas, l'interpréteur de Ruby est écrit en C, ce qui permet aussi d'écrire du code C dans une application Ruby si le besoin s'en fait sentir.
Et même si on a déjà parlé de Rails, on peut aussi mentionner Sinatra, un micro framework créé en Californie et soutenu par Heroku, GitHub et dont Travis CI s'occupe aujourd'hui.
Sinatra est un DSL qui permet de créer une application web en 4 lignes :

```ruby
require 'sinatra'

get '/' do
  'Hello world!' # un programme Hello World renvoyant 200 OK sur root avec Sinatra
end
```

Sinatra a été le second framework Ruby au succès massif, mais d'autres plus discrets ont aussi vu le jour et on dénombre quasiment une vingtaine de frameworks Ruby aujourd'hui. Et si certains sont spécialisés dans des fonctions très précises du backend, comme créer une API, aucun ne prétend être le meilleur en qualité du front et c'est encore une démonstration des qualités du langage : Ruby connaît ses forces et se concentre dessus.
Ruby est un très bon outil pour coder rapidement du code métier. Les principes de convention supérieure à la configuration de Rails nous permettent de générer des fichiers dédiés au rendu visuel dans le navigateur, mais il est très aisé d'intégrer un framework de front à une application Rails ou sous un autre framework Ruby, si bien que voir des stack Rails/Sinatra être complétées par Angular JS, Vue JS, Node.js ou EmberJS pour ne citer qu'eux, est chose courante.
Un autre des points qui me fait autant aimer Ruby, c'est qu'il est lisible et concis. Tellement simple à déchiffrer que parfois on a l'impression de simplement lire de l'anglais

```ruby
first_name = book.author.name.split.first if book.author_id.present? && book.author.name.present?
```

Cette ligne de code renvoie le prénom (first name) de l'auteur et se lit presque comme une phrase en anglais. Elle est aussi très courte et efficace en ce sens qu'elle permet d'évaluer une condition et d'exécuter une opération conditionnelle en une seule ligne tout en restant claire quant à son utilité. Pour faire encore plus court on pourrait supprimer les `present?`, mais la méthode présente un avantage que nous détaillerons juste après.
On voit aussi ici l'utilisation de l'enchaînement de méthodes sur un objet, très pratique pour effectuer des opérations complexes en chaîne sur les objets. Et rappelez-vous, en Ruby, tout est objet !
Et pourquoi c'est cool que tout soit objet ? Chaque objet à une classe qui descend elle-même d'une classe Ruby ce qui lui donne tout un tas de méthodes utilisables comme la méthode `split` que nous avons utilisé sur la string `name` et la méthode `first` que nous avons utilisé sur le tableau - array en Ruby - obtenu en résultat du `split`.
Ce qui est fou c'est que si une méthode semble logique sur un type d'objet, comme `first` qui retourne l'élément à l'index 0 d'un array, et si elle semble logique sur un autre type d'objet, elle fonctionnera également sur cet autre type d'objet.
Par exemple, si nous avions utilisé `first` directement sur la string contenant le nom de l'auteur, il nous en aurait retourné la première lettre. Et c'est un comportement encore plus logique quand on pense au fonctionnement du C, pour qui une chaîne de caractères est un tableau de `char`. Et `char` est un type capable de contenir une seule lettre. `first` nous retourne donc le `char` situé à l'index 0 de la chaîne de caractères `author_name`.

v	i	c	t	o	r		h	u	g	o
0	1	2	3	4	5	6	7	8	9	10

Le fait d'utiliser la méthode `present?` montre également l'un des autres avantages de la syntaxe Ruby. Une méthode se terminant par un `?` retournera *obligatoirement* un booléen `true` ou `false`, rendant prédictible le type de données à évaluer tout en rendant explicite le fait que l'on se pose une question sur l'objet. Ruby utilise aussi un symbole `!` pour les

méthodes opérant des modifications sur l'objet qu'elles manipulent, attirant ainsi l'attention de la personne qui lit ce code sur le caractère irréversible de l'action.
De la même manière le scope des variables est toujours explicité par un symbole rendant l'information lisible mais aussi le développeur conscient de ses choix quant à sa façon de traîter la donnée :

- Un nom de constance commencera toujours par une Majuscule
- Une variable globale sera précédée d'un $
- Une variable d'instance sera symbolisée par un @
- Une variable de classe quant à elle sera représentée par deux @@

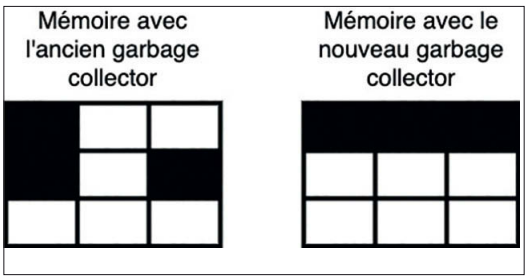
De par son adoption par le plus grand nombre, et en s'imposant comme un langage avec lequel il faut compter, Ruby a poussé les plus gros fournisseurs de software as a service à créer leurs propres gems pour rendre leur services disponibles sous un format que les développeurs Ruby aiment : les Gems. Comme je vous l'ai montré plus haut avec l'exemple de l'envoi de SMS utilisant la gem Twilio, les plus gros proposent tous leurs services sous ce format simple à intégrer.

Tous les plus gros acteurs du cloud proposent également l'hébergement de code Ruby, et mettre en production un projet Ruby se fait sans accroche sur toutes les plateformes.

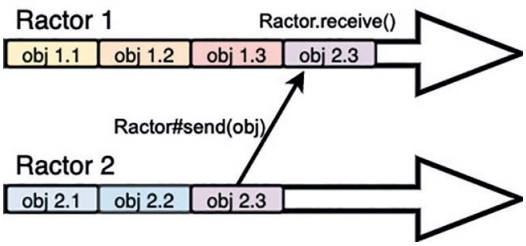
Tous ces points, sans compter les améliorations et stabilisations font de Ruby un langage mature, à l'écoute des usages des développeurs et des besoins du marché et pour moi l'un des plus sympas avec lesquels coder depuis quelques années. Cependant tout n'a pas toujours été tout rose dans le développement Ruby et la lenteur du langage a souvent été l'un des points sur lequel il s'est fait descendre par ses détracteurs. Mais une écoute constante de la communauté de la part des mainteneurs du langage ont permis d'apporter des améliorations continues à cette problématique pour toucher le point d'orgue en décembre 2020. En effet, le 25 décembre, comme un cadeau à la communauté, Ruby 3.0.0 est sorti, et sa promesse était d'être trois fois plus rapide que sa version majeure précédente.
Alors promesse tenue ou non ? Sans plus de suspense, oui ! Ruby 3.0.0, surnommé Ruby 3x3, est trois fois plus rapide que Ruby 2.x.

Mais ça ne s'est pas fait en un jour, puisque que ce sont plus de sept ans d'améliorations constantes qui séparent les deux versions. Les améliorations ont été implémentées progressivement, et en réalité, si Ruby 3x3 à fait autant de bruit, c'est pour ses nouvelles fonctionnalités.
Cette version affiche des objectifs clairs : être plus rapide et améliorer la simultanéité.

Etre plus rapide: La version 2.6 du langage a vu apparaître l'option du MJIT, un compilateur JIT - Just In Time - basé sur les Méthodes. En clair, ce que ça veut dire c'est que le programme est capable de détecter si des méthodes spécifiques du code sont appelées de façon répétitive pendant l'exécution. Si de telles méthodes sont détectées, elles sont placées dans une file d'attente puis sont converties en code natif - compilées - par le compilateur.
Cette version voit aussi venir l'apparition de la compaction automatique dans le garbage collector. Ce dernier est désormais capable de réorganiser les objets dans l'espace mémoire alloué à l'application afin de les grouper, laissant ainsi de grands emplacements mémoire continus disponibles au lieu de plusieurs petits espaces éparpillés dans la stack. De cette façon, écrire de nouveaux objets, même gourmands en mémoire, sera plus rapide.



Une meilleure simultanéité : Ractors est la nouvelle abstraction concurrente de type Acteur-Modèle intégrée à Ruby 3.x3. Elle a été conçue pour la thread-safety.
Son fonctionnement est simple: vous pouvez déclarer plusieurs Ractors, ils s'exécuteront en parallèle. Cependant ils ne partageront pas d'informations entre eux, la plupart de leurs objets étant non partageables. Un système de messages permet de synchroniser l'exécution. Il faudra alors explicitement envoyer `Ractor#send(obj)` et recevoir `Ractor.receive()` les messages dans les Ractors.



Falcon Rack vient aussi s'ajouter à cette version. Falcon Rack est un framework d'entrée/sortie asynchrone composable pour Ruby. Il permet au serveur d'accepter et de traiter d'autres requêtes en attendant la base de données, la mémoire ou une requête API et il vient par défaut dans la version 3.

Collaboration
Même s'il est facile de développer localement une application Ruby grâce au Gemfile, il est préférable pour une équipe travaillant sur le même projet d'avoir le même environnement de développement pour tous. Vagrant est très souvent utilisé dans le développement d'applications Ruby on Rails, car il permet grâce à une machine virtuelle d'émuler l'environnement de production ou tout autre environnement et de partager cette configuration avec d 'autres personnes grâce à un Vagrantfile et un Cheffile. Ces deux fichiers peuvent être comparés à un Dockerfile et un Gemfile pour cet environnement.

Production

Il est facile de mettre une application Ruby en production. La grande majorité des hébergeurs supportent le langage et mettent à jour régulièrement les versions prises en charge sur leurs plateformes. Il est tout à fait possible également d'utiliser Docker, avec ou sans Kubernetes.

En général, et surtout si l'on déploie une application Rails, on la couple à une ou plusieurs bases de données. Bien que Ruby On Rails embarque SQLite dans son mode de développement, il est en général préférable de travailler avec une ou plusieurs PostgreSQL. Il est également courant de retrouver une instance Redis compagnonne de l'application.

Pour mettre l'application en production dans le cloud sans Docker, il faudra fournir à votre instance dans le cloud les variables d'environnement nécessaires au bon fonctionnement de l'application, sans oublier de spécifier la version de Ruby utilisée.Vous aurez aussi besoin d'un secret, générable localement avec `rake secret`.

Il faudra également avoir tous les fichiers de migration de base de données dans le répertoire `db/migrate` et s'assurer que vous n'avez pas utilisé de fonction de migration spécifique à SQLite mais uniquement des fonctions compatibles avec votre base de données de production - qui sera probablement une PostgreSQL, auquel cas vous devrez aussi ajouter la gem `pg` à votre Gemfile.

Assurez-vous de lancer au moins une fois la commande `bundle install` localement, ça créera le fichier `Gemfile.lock`, sur lequel la plupart des fournisseurs de cloud se baseront pour gérer les versions de vos dépendances.

N'oubliez pas de spécifier dans votre fichier `config/environments/production` un ou plusieurs hôtes autorisés comme suit :

```
config.hosts << "monurl.com"  # ajout d'un hôte unique
config.hosts << /[a-z0-9]+\.monurl\.com/ # ajout de sous domaines dynamiques à l'aide d'une regex
```

Une fois cela fait, les différents fournisseurs de solutions cloud ont des instructions spécifiques à chacun mais il est certain que vous devrez instruire le lancement de la commande `rake db:migrate` afin de modeler la base de données de production à votre besoin. Vous disposez également de la commande `rake db:seed` qui permet de peupler la base de données avec des datas créés par vous dans le fichier `db/seeds.rb` de votre application.Si votre application embarque un front-end ou des modules Javascript, vous aurez besoin de lancer

```
rake assets:precompile
bin/webpack
```

Et de définir la variable d'environnement `NODE_ENV=production`.

Il est à noter que Rails 7 - qui n'a pas encore de date de sortie officielle - prévoit de "déprécier doucement" webpack pour la gestion du JavaScript, ce qui signifie que Webpacker sera toujours présent mais ne sera plus défini par défaut, au profit de "import maps", qui est un outil dans le navigateur qui vous permet de mapper un nom logique à un module téléchargé directement dans le navigateur sans avoir besoin de gérer le bundling côté serveur, et un wrapper Rails pour gérer ce mapping depuis le code serveur. Tout ceci étant encore en cours de développement, n'oublions pas de garder un oeil dessus dans les mois à venir !

Puis votre fournisseur de cloud lancera sûrement la commande `rails s` ou équivalent qui lance le serveur, *and voilà* vous serez en production, merci la magie du cloud pour les devs qui ne font pas de devops !

On peut aussi noter que coder un MVP avec Rails est en moyenne 30% plus rapide qu'avec n'importe quel autre framework, ce qui en fait le framework le plus rapide en termes de time-to-market. Le framework est aussi compatible avec la création de progressive web apps, et certaines Gems promettent aussi de transformer votre application Rails en code natif iOS et Android. Mais comme pour toute application mobile, une connaissance moyenne du code Swift et Java sera nécessaire pour obtenir une application mobile parfaitement adaptée à chaque plateforme.

Bref, quel que soit ce que l'on veut faire, c'est possible avec Ruby. Reste à déterminer si c'est le meilleur choix technique pour votre cas d'usage, et même si j'aime Ruby d'un amour sans failles, vous savez aussi bien que moi que ce ne sera pas toujours le cas. Notre amour d'une technologie ne doit pas nous empêcher de rester objectifs quant à ses possibilités.

Travailler dans l'écosystème Ruby en 2022

Sans surprise, puisqu'il s'agit d'un poste lié au développement, l'offre est constante et variée. Le marché étant en recherche perpétuelle d'encore et toujours plus de devs, les fiches de poste disponibles contenant du Ruby dans la stack ne sont pas rares.

D'ailleurs il suffit en général de mentionner sur son profil linkedin une première expérience professionnelle avec le langage pour que votre messagerie se remplisse d'in-mails de recruteurs qui recherchent des personnes ayant des compétences dans la techno. Ruby a aussi été adopté par les plus grands et ces compagnies recrutent régulièrement à des postes techniques.

En travaillant avec Ruby, vous serez soit :

- Dev 100% backend auquel cas vous coderez des applications server-side, implémentez de la logique métier et gérerez les opérations liées aux bases de données
- Dev fullstack, ce qui veut dire que vous ferez la même chose que vos homologues backend en y ajoutant la connaissance de frameworks de front, bien souvent basés sur Javascript tels que Angular ou Vue.

D'après le classement ChooseYourBoss, les salaires en France seraient actuellement répartis de la façon suivante :

	Junior	Expérimenté	Sénior
Années d'expérience	0-1	2-5	6+
Salaire en euros	35-40K	40-50K	60-65K

Vous pouvez aussi devenir CTO d'une startup ou d'une boîte, avec des salaires avoisinant les 100K en France.
Ceci dit, c'est vers l'étranger qu'il faut se tourner pour être le mieux rémunéré et à titre de comparaison à l'international, un sénior gagne jusqu'à 120K, mais cette fois en dollars et des différences à la hausse sont aussi visibles sur les postes requérant moins d'expérience.
De fait, aller plus loin que la lecture de la documentation dans l'apprentissage de la langue de Shakespeare peut vous être très bénéfique, puisqu'il faut indispensablement parler anglais couramment pour pouvoir prétendre à ces postes tout en bossant tranquillement depuis son salon en France.
Un bon compromis peut être de travailler pour une filiale française de boîte technologique étrangère. C'est idéal si l'on veut aller au bureau et voir les collègues tout en bénéficiant d'un salaire souvent plus intéressant que dans une boîte franco-française.
En France, tous types d'entreprises recrutent et vous devriez être en mesure de trouver l'équipe qui vous convient que ce soit dans une petite structure de moins de 20 personnes ou dans une boîte leader de son secteur ayant plus de 500 employés. Il faut toutefois noter que les postes à pourvoir restent concentrés dans les grosses villes, Paris en tête, mais on peut aussi citer Lyon, Lille, Nantes, Toulouse ou Cannes si vous préférez le soleil.
La tendance au full remote observée depuis le début de la pandémie de la Covid 19 se retrouve aussi dans les offres d'emploi Ruby et si vous préférez dire adieu à jamais au bureau, c'est également possible.
Et le freelance ? C'est possible quand on fait du Ruby ? Vous vous en doutez, la réponse est oui !
Avec des missions courtes allant de trois à six mois en général, il est tout à fait possible de trouver des missions freelance, sur site ou à distance. Les clients de Shopify sont aussi un vivier d'entrepreneurs se lançant dans l'e-commerce sans avoir forcément les compétences techniques pour faire évoluer leur site au-delà des possibilités no code offertes par l'outil et ce public est aussi demandeur de missions ponctuelles.
Niveau argent, le taux journalier moyen se situera en moyenne entre 300 et 650 euros par jour en fonction du niveau d'expérience. Il faudra bien sûr payer les taxes sur ce que l'on facture.
En termes de structure juridique, les devs freelance choisissent en général l'auto entreprise car très simple de gestion mais ses plafonds font qu'à partir d'un certain niveau de revenus il est nécessaire de se tourner vers d'autres types de structures. À cet effet, l'Entreprise Individuelle, l'EURL et la SARL sont les choix les plus plébiscités.

Ruby vers le futur

Ruby nous prépare plein de bonnes choses pour cette année mais avant de plonger dans un bain de bonnes nouvelles pour les adeptes du langage, répondons à une question que les intrigués pourraient se poser : est-ce que ça vaut encore le coup d'apprendre Ruby en 2022 ?
Ma réponse est bien évidemment !
Reboosté depuis la version 3.0.0, Ruby connaît un regain d'intérêt sans jamais avoir disparu du paysage et prouve que c'est un langage avec lequel il faudra encore compter pour les années à venir. Et qui ne veut pas apprendre un langage conçu pour le bien-être et le bonheur des devs ?
Si vous lisez l'anglais je vous recommande le livre Head First Ruby : A Brain-Friendly Guide, qui explique de manière visuelle beaucoup de concepts du langage.
En termes de frameworks, même si Rails, pour les fullstack, et Sinatra, pour les micro frameworks, dominent largement et restent les meilleurs choix pour trouver facilement du travail ou des projets sur lesquels collaborer, d'autres options s'offrent à vous.
Hanami fait parler un peu de lui chaque année, car il consomme très peu de mémoire, à peu près 60% de moins que Rails. Grape quant à lui est un framework pour API REST-like qui vient complémenter les applications existantes.
On peut aussi mentionner Padrino, un framework full stack construit sur Sinatra qui a pour ambition de vous accompagner sur tous vos types de projets, que ce soit un simple JSON web service ou une application web complète.
Connaître la base du Ruby vous aidera à vous sentir à l'aise rapidement sur n'importe lequel de ces frameworks grâce à la simplicité apparente du langage alors n'oubliez pas que votre choix doit prendre en compte le type de projet que vous souhaitez réaliser et les possibilités offertes par le framework.
Quelques incertitudes : Ruby 3x3 introduit une fonctionnalité optionnelle de vérificateur de type statique. C'est un pas vers un typage fort du langage mais qui pourrait bien rester optionnel car la core team ne semble pas vouloir forcer les déclarations de type. Malgré la surprise face à cette annonce, la communauté sait par habitude que les features présentées comme optionnelles ont tendance à devenir la norme dans les versions suivantes de Ruby. Il faudra donc garder l'œil ouvert dans les prochains mois ou années et suivre si des décisions qui vont en ce sens sont prises.
Toutefois, le 25 décembre 2021 sortait la release de Ruby 3.1.0 et cette version poursuivra les efforts d'amélioration du langage, stabilisant l'existant dans un premier temps. Il ne devrait donc pas y avoir de bouleversement majeur cette année.
C'est du côté du framework Ruby on Rails que vont se produire beaucoup de changements. La version 7 du framework abandonne Webpacker, l'outil historique de bundling JS en Rails, et privilégie désormais une approche no-code pour gérer les packages NPM dont je vous ai parlé plus haut.
Adieu aussi Turbolinks et Rails UJS, faites place à Stimulus JS pour gérer le javascript frontal.
Active Record, l'ORM embarque de nouvelles capacités de chiffrement pendant l'exécution, et des fonctionnalités supplémentaires pour gérer l'asynchrone ont été ajoutées aussi bien pour les requêtes que pour les tests.

Je peux affirmer avec certitude que je continuerai à coder en Ruby en 2022, et pour les années à venir. Le langage est robuste et plaisant, la core team ne laisse pas tomber le projet et le pousse toujours à évoluer en fonction des usages et des avancées technologiques.

Et vous, je vous ai convaincu ?

Nathalie Seitz

Ingénieure chez Simplicité Software

De formation scientifique, j'ai démarré ma carrière en tant qu'ingénieure logicielle. Quand j'ai découvert la plateforme Simplicité, j'ai tout de suite adhéré au concept de ne pas réinventer la roue à chaque application et de se concentrer sur l'essentiel du métier de l'utilisateur pour sa plus grande satisfaction.

Exemple d'une application de réservation de chambre d'hôtel

Le développeur, également appelé designer dans Simplicité, va commencer par paramétrer ses objets et les relier entre eux grâce au modeleur graphique, dessiner ses écrans grâce à un éditeur de template en glisser/déposer, définir les différents droits des utilisateurs et enfin coder les règles de gestion et les composants spécifiques, au sein de l'éditeur de code de la plateforme ou via son propre IDE connecté à la plateforme via GIT.

Le moteur de la plateforme restitue à la demande le paramétrage et exécute le code pour fournir en temps réel les interfaces UI ou API.

Voici un exemple d'utilisation sur un cas simple pour en comprendre les concepts. Nous allons voir les différentes étapes pour construire notre application de réservation : installation, design, test, livraison, insertion de données et exploitation.

Installation d'une plateforme Simplicité vierge

Il suffit de télécharger une image Docker dans la version de son choix (registry.simplicite.io). Une image docker contient :

- un serveur Tomcat et son (open)JDK ou JRE,
- une base de données dans laquelle est pré-stockée le paramétrage système (par défaut HSQLDB pour les dev, puis on peut choisir PostgreSQL, MySQL/MariaDB, Oracle ou SQL Server pour la production) et son driver JDBC,
- et enfin le runtime Simplicité composé de libs java et d'une webapp.

Une fois l'installation effectuée le design peut commencer :

La première étape consiste à fabriquer notre modèle logique en nous servant du modeleur qui permet de créer les objets, leurs attributs et les relations qui vont composer notre application :

- Un menu hôtel qui contient : des "Chambres", "Types de chambre", "Services", "Équipements", "Moyens de paiement".
- Un menu de gestion des réservations qui contient : des "Clients", un processus pour "Faire une réservation", des "Demandes de réservation" de différents statuts (en attente, traitée, clôturée), un objet "Paiements", "Accompagnants" et "Réservations". **Figure 1**

Par exemple, lors de la réservation d'une chambre d'hôtel nous avons besoin d'avoir les informations suivantes :

- Descriptif : Libellé, Superficie, Photo de la chambre, Tarifs,
- Type de chambre : simple, double, triple,
- Services afférents : Parking, Wi-Fi gratuit, Climatisation, TV... et Sèche cheveux,
- Équipements : Penderie, Gel hydroalcoolique, Minibar, Coffre-fort, Bureau, Téléphone… et pourquoi pas Matériel de repassage,
- Moyens de paiement du client : Paiement direct à l'hôtel, CB, Virement bancaire, PayPal, etc.

Grâce à l'outil de création par glisser-déposer, nous fabriquons les formulaires avec les différents éléments en atelier avec nos chers utilisateurs. Nous réalisons la même opération pour la gestion des réservations en y ajoutant un processus de création de Demande de réservations à l'aide d'un diagramme d'états : En attente > Traitée > Clôturée. **Figures 2 et 3**

L'application est accessible également sur tablette ou mobile, car les écrans sont nativement responsives : **Figure 4**

Nous créons également un processus avec 4 étapes :

1. Saisie des informations du client : coordonnées + adresse,
2. Demande de réservation en sélectionnant la période de réservation, le nombre de chambres et les options,
3. Possibilité de rajouter des accompagnants si besoin,
4. Puis finalisation pour créer la réservation.

Dans notre exemple, cela donnera à l'étape 2 : **Figure 5**

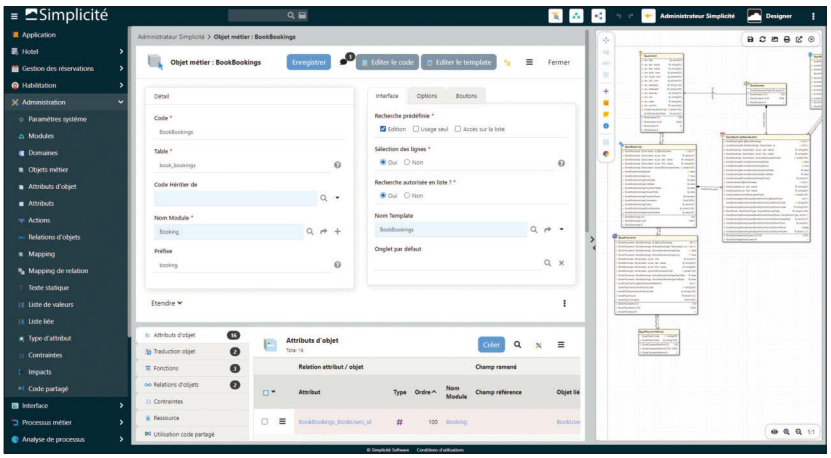
Nous pouvons également très simplement :

- Créer des tableaux croisés et des graphiques basés sur les axes des réservations (quantité, chiffre d'affaires par chambre, par mois...), **Figures 6 et 7**
- Créer une vue Kanban ou un Calendrier pour voir les différentes réservations par statut et les déplacer par glisser/déposer pour changer les statuts ou les dates. **Figures 8 et 9**

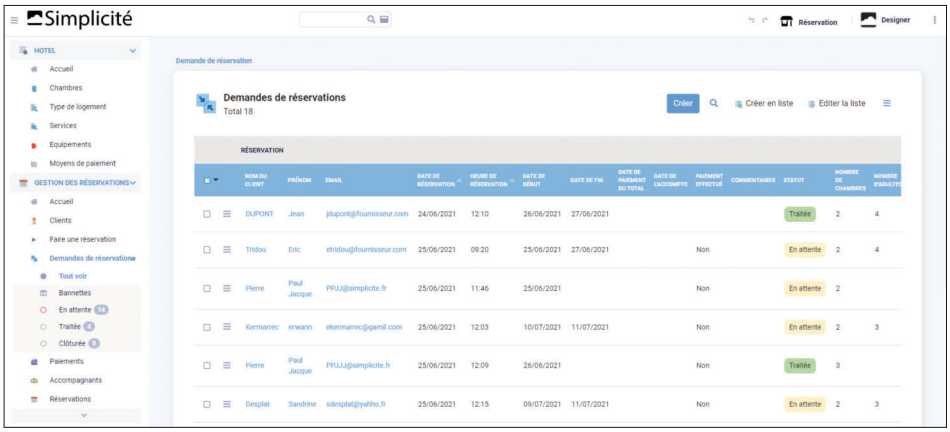
Un peu de code pour notre application low-code

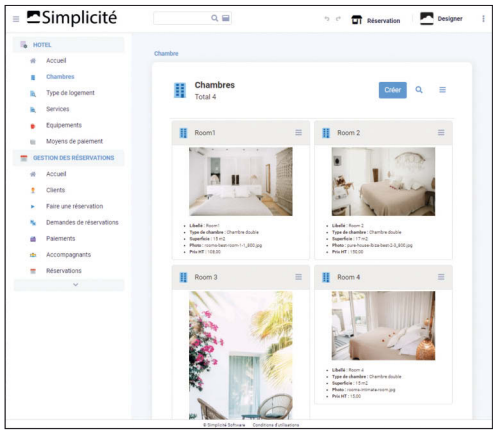
L'application est finalisée à 80%, jusqu'ici nous n'avons pas eu besoin d'écrire une seule ligne de code, mais imaginons que nous souhaitions ajouter une règle de gestion sur la date de début de la réservation qui doit être antérieure à la date de fin : on va utiliser les "hooks" des objets et développer en JAVA. Lorsqu'un objet ou un processus est configuré, il a un

Figure 1



Figures 2 et 3





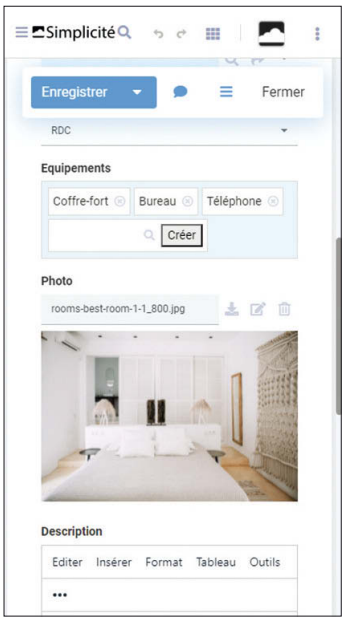


Figure 4

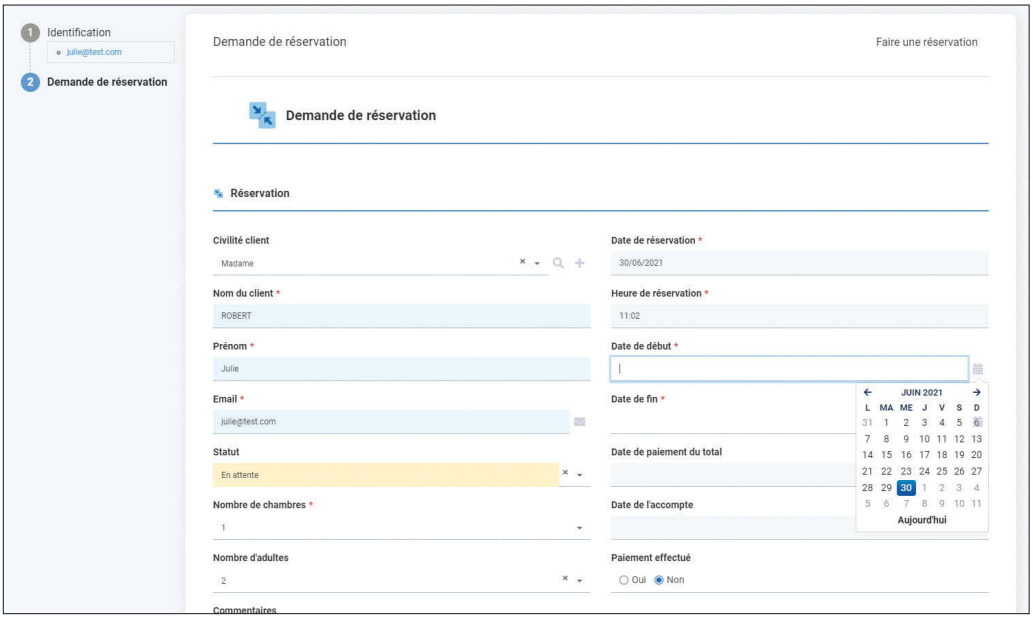


Figure 5

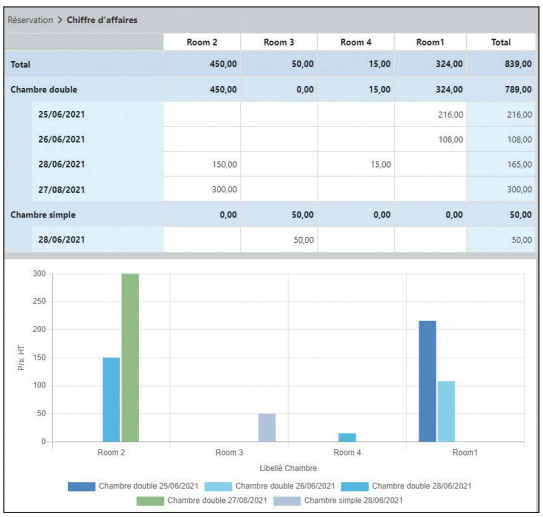


Figure 6

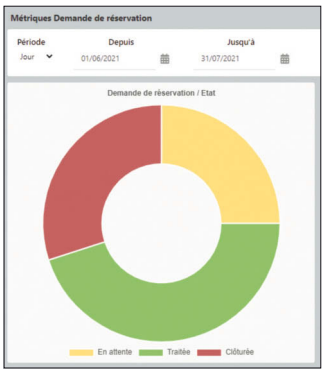


Figure 7

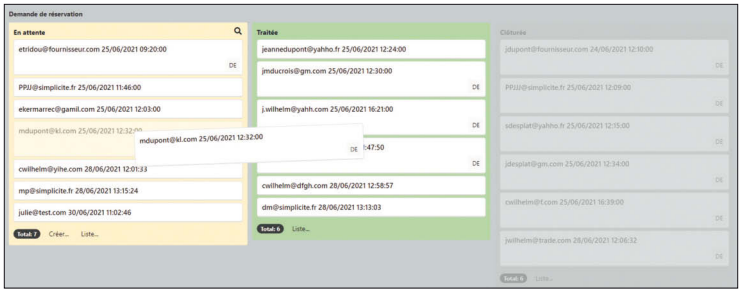


Figure 8

comportement par défaut sur la base des différentes propriétés configurées. Son instance en mémoire hérite du comportement standard, et il est alors possible d'étendre ou modifier son comportement en surchargeant le cycle de vie de l'objet via ses hooks et en fonction des droits/profils de l'utilisateur : quand l'objet est chargé en session, quand on valide la saisie, quand on initialise la liste, quand on change une valeur en front... On ajoute le code Java (en back) ou JavaScript (en front) au sein de l'éditeur de code intégré ou depuis son IDE préféré.

Dans notre exemple, nous allons utiliser le hook postValidate héritée de ObjectDB. On surcharge le cycle de vie de la demande de réservation après que la plateforme ait validé le paramétrage effectué (type de champs, champs obligatoires, etc.). Dans le hook postCreate (une fois la demande de réservation créée), on envoie un mail au client pour le notifier de la prise en compte de sa réservation.

Si vous avez placé votre module sous GIT, il deviendra un projet apache Maven dans votre IDE et vous permettra également de faire du remote-debug, de la complétion avec accès à la Javadoc.

```
// Back-end code for object BookBookings
public class BookBookings extends ObjectDB {
 private static final long serialVersionUID = 1L;
```

```
/** Hook override: error if the start date is greater than the end date */
@Override
public List<String> postValidate() {
  List<String> msgs = new ArrayList<>();
  if (Tool.diffDate(
    getFieldValue("bookBookbookingsStartDate"),
    getFieldValue("bookBookbookingsEndDate")) <=0)
    msgs.add(Message.formatError("BOOK_ERROR_DATE",null,"bookBook
bookingsEndDate"));
  return msgs; }
/** Hook override: send notification to customer after booking creation */
@Override
public String postCreate() {
  try {
    MailTool mail = new MailTool(getGrant());
    mail.addRcpt(getFieldValue("bookBookusersEmail"));
    mail.setSubject("Demande de réservation reçue / Booking creation");
    // Get the email template and substitute data
    String body = HTMLTool.getResourceHTMLContent(this, "TEMPLATE_MAIL")
      .replace("[TITLE]", getFieldValue("usr_title"))
      .replace("[NAME]", getFieldValue("usr_last_name"))
      .replace("[FIRTSNAME]", getFieldValue("usr_first_name"))
      .replace("[DATE]", getFieldValue("bookBookbookingsDate"))
      .replace("[HOUR]", getFieldValue("bookBookbookingsHour"));
    mail.setBody(body);
    mail.send();
  } catch (Exception e) {
    AppLog.warning("Error sending notification to customer", e, getGrant());
  }
  return super.postCreate();
 }
}
```

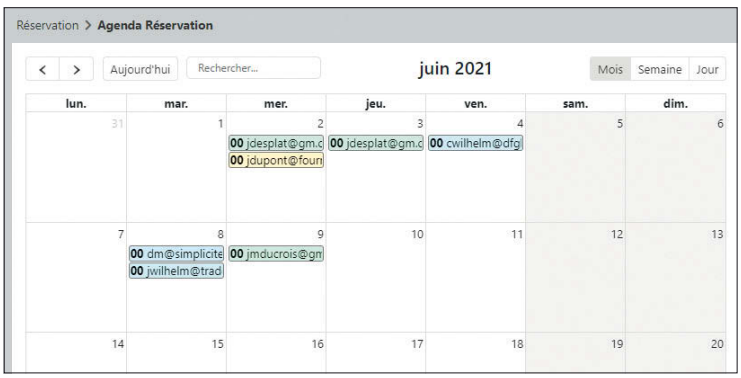


Figure 9

Figure 10

Autre exemple de JavaScript front pour afficher la géolocalisation du client en utilisant les API Google Map : **figure 10** et qui permet d'ajouter un composant UI sur le formulaire du client : **figure 11**

Tests unitaires

Il faut distinguer 3 niveaux de tests :

1 Pour la partie back en Java/SQL : vous pouvez implémenter vos classes JUnit depuis votre IDE ou directement dans Simplicité. Une action de l'objet vous permet de lancer les tests unitaires par l'interface ou par script (curl). On peut également brancher un outil de test des performances par montée en charge pour tester les API (type apache JMeter).

```
/** JUnit test class */
public static class BookBookingsTest {
   @Test
   public void testDiffDate() {
      assertTrue(Tool.compareDate("2021-06-11", "2021-06-01")>0);
      assertFalse(Tool.compareDate("2021-06-01", "2021-06-11")>0);
   }
}
/** Hook override: launch JUnit tests classes */
@Override
public String unitTests() {
   JUnitTool t = new JUnitTool(getGrant());
   return t.run(BookBookingsTest.class);
}
```

2 Pour la partie paramétrage : Simplicité intègre un audit du design permettant de lever des alertes sous la forme de posts sociaux collaboratifs pour les développeurs sur chaque objet dont le paramétrage ne semble pas conforme (ex : un attribut non utilisé, une liste de valeurs vide, un objet métier sans clé fonctionnelle ou encore une runtime exception lors de l'exécution du code ajouté). Simplicité corrige également votre paramétrage en vous le signalant s'il détecte une incohérence.

3 Pour la partie front UI : rien à tester, car tout fonctionne nativement. Si vous avez développé du spécifique en front-end, vous pouvez connecter n'importe quel outil de tests du marché (comme Selenium ou Jasmin).

Automatisation des tests : dans une prochaine version, Simplicité proposera son propre outil d'enregistrement de scénario métier (ex : je déroule une réservation de chambre pour 3 personnes dont une est allergique à la moquette rouge et au gluten) afin d'en faire une base de cas de tests rejouables automatiquement pour assurer la non-régression à chaque montée de version, ou encore générer l'assistant "étape par étape" pour les utilisateurs.

Installation d'une application

La plateforme low-code permet d'exécuter des **modules** Simplicité, chaque module contient les livrables :

- le paramétrage : tous les objets de configuration (objets, champs, menus, processus, habilitations…) sont exportés sous forme de fichier xml ou Json,
- les fichiers : sources java/JavaScript et ressources HTML, css/less, images, pdf, etc.

- et optionnellement des jeux de données d'initialisation (les équipements et les caractéristiques des chambres par exemple).

Les développeurs fabriquent leur application sur une instance de développement, et exportent leurs modules pour les installer sur une instance de qualification puis de production. L'installation d'une application se déroule donc en deux étapes :

1. installation d'une plateforme Simplicité vierge
2. import du ou des **modules** de configuration Simplicité constituant l'application métier.

L'import des modules applicatifs peut se faire de différente façon suivant votre façon de travailler :

- manuellement par upload de l'archive via la UI en tant qu'administrateur,
- via un "pull" depuis un repository GIT privé ou public (GitHub),
- par commande : "curl" à distance, ou via outil de déploiement (type Jenkins),
- ou encore depuis un **App-Store** Simplicité (public ou privé) en un clic : **Figure 12**

Insertion de données de référence

Enfin, si on a configuré les ordres d'export des objets de son modèle, la plateforme génère un jeu de données qu'il est possible d'installer sur une nouvelle instance (environnement de recette ou de production) tout en utilisant la supervision des imports pour vérification des éventuels rejets : **Figure 13**

Le langage du jeu de données est structuré (XML, JSON ou YAML). Dans notre exemple, on peut exporter une liste de chambres pour les réimporter dans n'importe quelle autre instance, donc, quelle que soit la base cible et en respectant tous les hooks qui auraient pu être ajoutés à l'objet pour contrôler ou réaliser des actions en cascade : **Figure 14**

Services d'exploitation :

Une fois notre application en production, il est possible de la superviser via le monitoring de l'instance :

- l'état général de la mémoire, les sessions en cours,
- les performances des services back (réponse des API) et front (affichage de la UI),
- les accès à la base de données et les requêtes les plus longues,
- les threads en cours, leurs stack et les class-loaders chargés,
- les user-agents utilisés.

Ces métriques permettent au développeur d'améliorer rapidement l'expérience utilisateur en optimisant son paramétrage, son code ou ses requêtes.

Conclusion

Contrairement aux idées reçues, les plateformes low-code ne sont pas uniquement des usines à faire des "petites" applications. Avec Simplicité, il est possible de développer de puissantes applications stratégiques, complexes et pérennes avec une large autonomie pour que le développeur puisse se concentrer sur les besoins utilisateurs à forte valeur ajoutée, et dont le code est généralement le plus intéressant à réaliser.

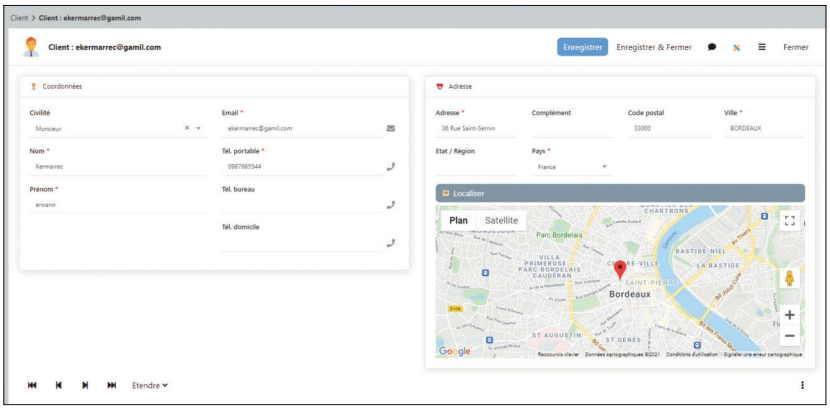


Figure 11

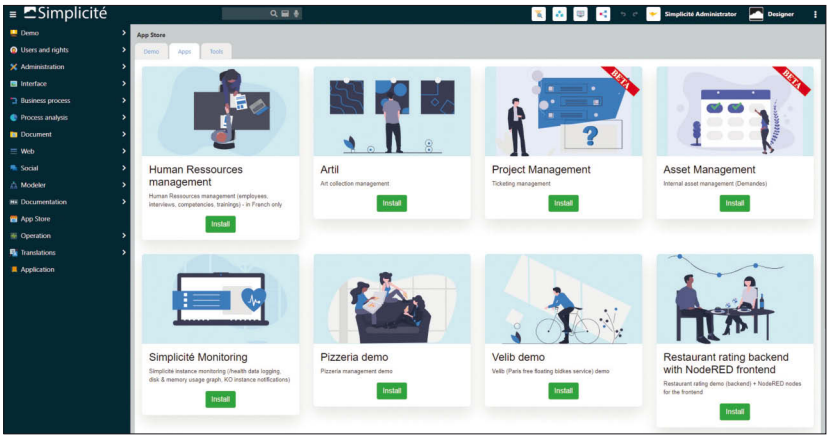


Figure 12

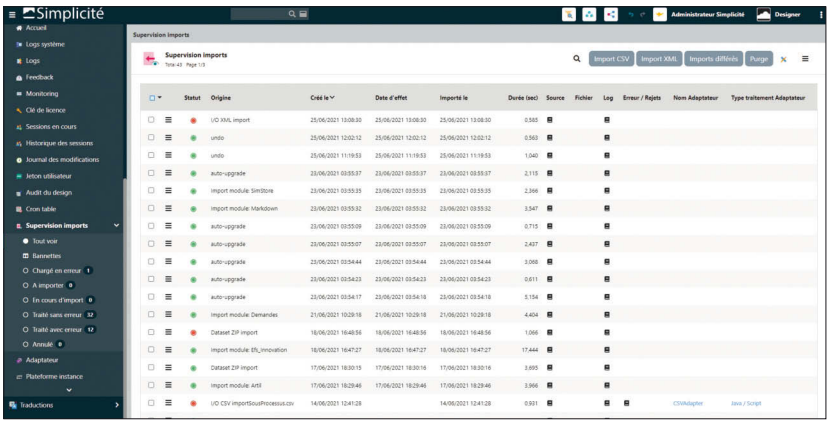


Figure 13

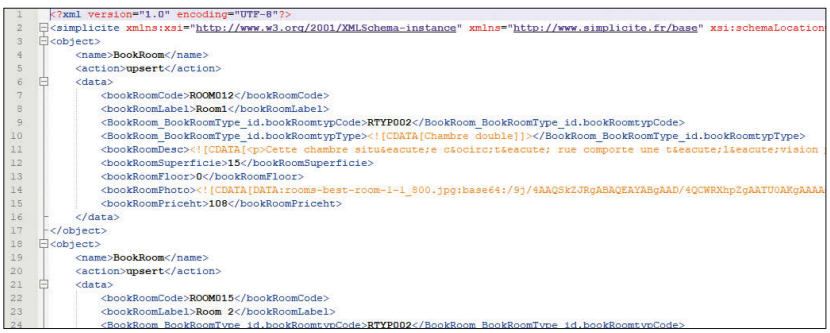


Figure 14

Ressources

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

Écrire une application en Flutter pour Android et iOS

Émilie C.

Ingénieur études et développement chez Blue Soft Group

Après une licence professionnelle spécialisée dans l'IoT Génie Logiciel Système d'Information et Objets Connectés à l'IUT Descartes, Émilie a intégré le groupe Blue Soft en 2018. D'abord dans le cadre d'une alternance, elle a fini par rejoindre le groupe en tant qu'Ingénieur études et développement.

Flutter est un SDK open source créé par Google et publié en version stable en décembre 2018. C'est un framework s'appuie sur langage Dart, développé par Google. Nous allons voir ici les avantages et inconvénients de Flutter, ses prérequis, les canaux git, la manière sur laquelle repose l'architecture du framework, comment créer son application avec les packages pub / stockage / architecture de l'application / pattern BLOC / widgets (statefull / stateless), l'internationalisation, le responsive, les best practices, la sécurité, la liste des commandes flutter ainsi que la manière de déployer son application.

Les promesses de Flutter sont le cross platform, un code initialement compatible avec Android et iOS qui se déploie aujourd'hui sur Windows, macOS, Linux et le web, et la possibilité de créer de belles applications avec peu code.

Les avantages et les inconvénients

Avantages :

- Open source : Flutter avance en permanence ce qui donne à tous la possibilité d'intervenir avec des pull request sur le repo.
- Dynamisme : une communauté très vive, omniprésente, propice au développement des idées. Cet engouement communautaire permet d'avoir des packages en permanence.
- Simplicité : tout le monde peut comprendre et écrire une belle application Flutter. On peut venir de n'importe quelle technologie, native ou non, back ou front, écrire en Flutter est simple notamment pour réaliser des IHM sobres et user friendly.
- Hot reload : on peut modifier et voir instantanément le résultat, sans avoir à redémarrer toute la compilation. C'est un gain de temps considérable en termes de développement.
- Performance : Flutter a un moteur graphique qui fonctionne comme pour les jeux vidéo. Il va dessiner les vecteurs des widgets et les traduire en langage natif. Seul le rendu final sera affiché, le rendant ainsi plus performant, plus rapide et plus léger que les autres langages cross platform.

Inconvénients :

- Cross Platform : lorsque l'on veut une application disponible sur plusieurs plateformes, il y a des limites logiques. Certains éléments et morceaux de code sont incompatibles entre plateformes : par exemple, la connexion internet ou le stockage (le cache d'un device n'est pas identique aux cookies, etc.).
- Taille du package : dans le cas de grosses applications, cela peut ralentir le lancement au démarrage.
- Performance : un langage cross platform ne peut pas égaler un langage natif. Cela demande une utilisation plus importante de CPU, de RAM ou de batterie pour atteindre le plus haut FPS qui est de 60.

Prérequis

Préparer son environnement de développement

- Installer Flutter https://flutter.dev/docs/get-started/install et choisir son environnement (ici Windows) (https://flutter.dev/docs/get-started/install/windows) :
  - Installer le SDK
  - Mettre à jour les variables d'environnements
  - Lancer la commande "Flutter Doctor" : manipulations à faire directement avec le Shell de Flutter, le CLI qui va indiquer tous les éléments qui manque
  - Selon sa plateforme (ici Windows) : Installer le SDK Android avec le lien proposé par Flutter dans la console
- Installer Java (JDK) et mettre le JAVA_HOME dans les variables d'environnements (possibilité d'outre passé ceci dans le cas où Android Studio est déjà installé via la commande « flutter config —android-studio-dir=[PATH_ANDROID_STUDIO_INSTALL] »

A noter *pour les étapes suivantes, même en développement à destination du web, Flutter a besoin de la chaîne d'outils d'Android. Il est donc nécessaire d'avoir l'IDE Android Studio.*

- Installer Android Studio (https://flutter.dev/docs/get-started/install/windows#android-setup)
  - Créer à minima 3 émulateurs : 5,6 pouces, 7 pouces et 10 pouces. Cela permet à l'application d'être construite de manière responsive, au fur et à mesure de son développement
- Lancer une nouvelle fois « flutter doctor » pour accepter les termes d'utilisation de la licence Android
- Choisir son IDE ou éditeur de code : VS Code remporte les suffrages en termes d'utilisation de ressources. Il ne lui restera plus qu'à installer l'extention « Flutter » via le market.

Les canaux

Dans Flutter, on retrouve 4 canaux par défaut :

- **Le canal master**, c'est la version la plus stable et la plus fonctionnelle,
- **Le canal dev** est en général fonctionnel, mais de mauvais

builds peuvent y passer, laissant un flutter instable. Il n'est pas recommandé, mais reste très intéressant à suivre pour voir les prochaines nouveautés et correctifs.

- **Le canal stable** est issu d'une branche bêta. Elle est recommandée pour les applis de production. Cette branche contient les hotfixs, c'est-à-dire flutter ios, Android et web.
- **Le canal bêta**, on y retrouve une nouvelle version chaque mois qui est stabilisée pendant quelques semaines et qui finit ensuite sur stable. Ce canal contient le flutter desktop.

A noter : *lorsque le desktop sera rajouté sur la branche stable, tous les projets déjà existants pourront le rajouter avec la commande « flutter create –platforms=windows,macos, linux .» lancée à la racine du projet. Il faudra peut-être se conformer à une version de SDK minimale, et à minima réaliser un « flutter upgrade » pour mettre à jour son environnement de développement.*

Overview de l'architecture

La technologie Flutter est conçue en couches, de la plus basse à la plus haute :

La couche « foundation » est la base du framework et la librairie essentielle qui fait le pont entre le framework flutter et l'engine.

La couche « rendering » est la couche abstraite qui permet de créer les arbres de rendus, c'est-à-dire la mise à jour automatique de l'arbre pour refléter les changements.

La couche « widgets » est la couche dans laquelle la programmation dite « réactive » commence avec la définition des classes réutilisables.

La couche « material » et « cupertino » comportent les librairies de panels de contrôles. Par exemple, une page, nativement, ne s'ouvre pas de la même manière entre Android et ios. C'est le theming.

Cette succession de couche est appelée le « Layer cake ».

La première fois que runApp() est appelé, plusieurs choses se produisent en arrière-plan :

- Flutter construit l'arborescence des widgets
- Il va parcourir l'arborescence des widgets et créer une deuxième arborescence qui contient les objets Element correspondants en appelant createElement() sur le widget
- Un troisième arbre est créé et rempli avec les RenderObjects appropriés qui sont créés par l'élément en appelant la méthode createRenderObject() sur le widget correspondant. Les renders objects contiennent toute la logique du rendu

A noter que le code Dart qui « paint » les visuels de Flutter est compilé en code natif qui utilise Skia en moteur de rendu graphique. C'est de là que Flutter détient sa rapidité d'affichage et son taux de FPS aussi élevé. Flutter intègre également sa propre copie de Skia dans le moteur, permettant ainsi au développeur de mettre à niveau son application pour rester à jour avec les dernières améliorations de performances, même si le téléphone n'a pas été mis à jour avec une nouvelle version Android.

Debug d'une application

Il existe plusieurs outils pour pouvoir débugger du code Flutter :

DevTools est une suite d'outils de performance et de profilage qui s'exécutent dans un navigateur et prend en charge diverses fonctionnalités :

- Le débug à la source
- L'inspecteur de widgets qui affiche l'arborescence des widgets et de leurs propriétés ainsi que le mode sélection de widgets
- Le graphique mémoire (memory profiler)
- La chronologie d'affichage de l'arbre des widgets
- Les logs

Android Studio/IntelliJ et VS Code, activés avec les plugins Flutter et Dart, prennent en charge un débogueur intégré au niveau de la source avec la possibilité de définir des points d'arrêt, de parcourir le code et d'examiner les valeurs, incluant le fameux hot reload.

« Flutter inspector » est un inspecteur de widgets disponible dans DevTools, et également depuis Android Studio et IntelliJ (activé avec le plugin Flutter). L'inspecteur vous permet d'examiner une représentation visuelle de l'arborescence des widgets, d'inspecter des widgets individuels et leurs valeurs de propriété, et d'activer la superposition de performances.

Rappel : un émulateur est très pratique, car il est disponible directement sur son ordinateur. Mais il peut ne pas refléter la réalité. Il est donc très important de toujours tester l'appli sur un vrai téléphone. Pour Android, il suffit d'activer le mode développeur et de brancher son téléphone à l'ordinateur. Il sera automatiquement détecté et proposé par VSCode.

Créer sa première application

Un template personnalisable est généré par défaut. Une ligne de commande est à lancer : « flutter create [nom_du_projet] » en snake case, et à cette ligne peut être rajouté une multitude d'options. Tout est configurable via les options du CLI de Flutter. Par défaut, le template comporte les fichiers pour Android (kotlin), iOS (swift) et web.

// **extrait console**

```
Microsoft Windows [Version 10.0.19042.1415]
(c) Microsoft Corporation. All rights reserved.

C:\Application>flutter create mon_projet
Creating project mon_projet...
 mon_projet\lib\main.dart (created)
 mon_projet\pubspec.yaml (created)
 mon_projet\README.md (created)
 [...]
 mon_projet\web\index.html (created)
 mon_projet\web\manifest.json (created)
Running "flutter pub get" in mon_projet...                 959ms
Wrote 81 files.

All done!
In order to run your application, type:

 $ cd mon_projet
```

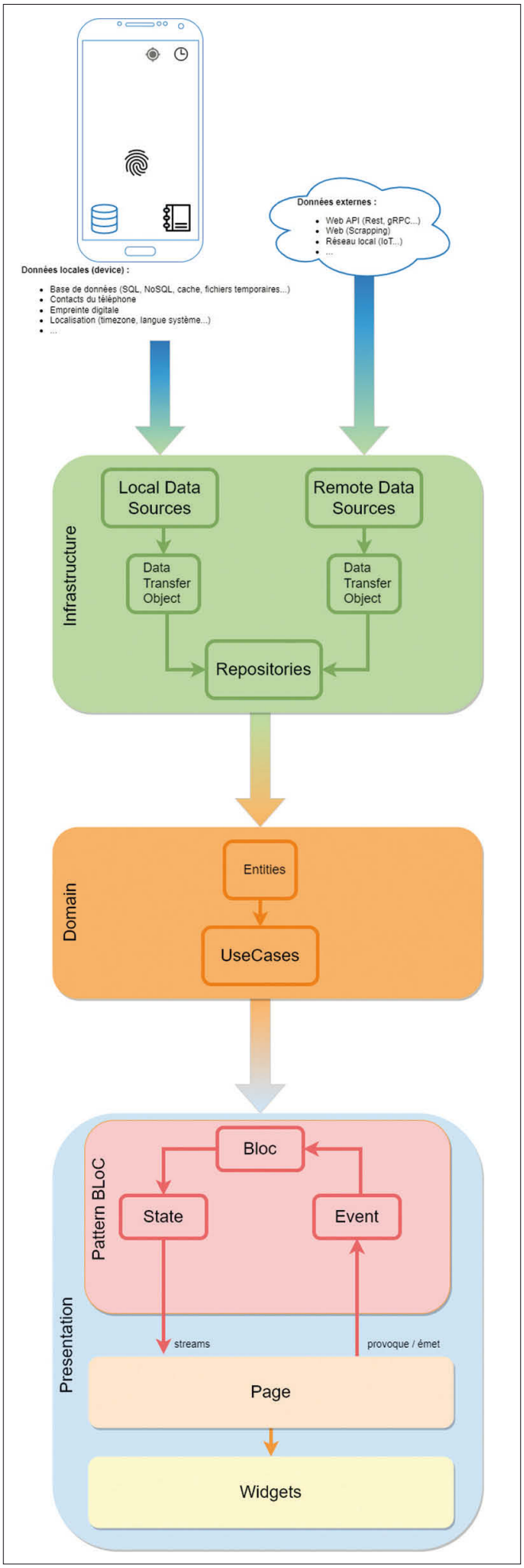


Figure 1

```
$ flutter run

Your application code is in mon_projet\lib\main.dart.
```

Comment choisir le bon stockage

On retrouve les 4 grandes familles de stockage :

- SharedPreferences : principe de key/value, option non secure mais rapide
- SecureStorage : principe de key/value, option secure (données sensibles) et rapide
- Tables structurées pour les données persistantes complexes : base de données Hive (noSQL - key/value) / SQLite (SQL), option plus lourde qui prend plus de temps
- Temporaire ou large objet simple/unique (exemple : fichier pdf) : utilise « path_provider » et permet d'utiliser le stockage temporaire natif (AppData pour Windows, le cache sous Android ou iOS, etc.)

Il ne faut jamais se restreindre à un seul type de stockage. Tous peuvent être adéquats à son contexte selon ses données et leurs utilisations.

Comment choisir : le bon stockage pub : https://pub.dev/

Pub est le moteur de recherche officiel des packages qui peuvent être utilisés et importés dans Dart et Flutter. Comme SDK, Pub est open source. Tout le monde peut publier des packages, Google y compris.

Pour éviter la déroute, attention de choisir le bon package dès le début :

- Toujours vérifier la compatibilité du package selon ses besoins de plateforme
- Pour des raisons de version, s'assurer que le package comporte le tag « null safety »
- Vérifier l'activité d'un repository pour éviter les packages oubliés
- Vérifier la qualité (code coverage si le code a été testé, documentations, nombre de téléchargements, etc.)
- Si plusieurs packages répondent à ces critères, toujours prendre celui qui est « mockable »

Architecture de son application

Il faut poser son architecture dès le début en s'aidant de schémas pour définir l'arborescence de son application. Ici nous choisissons la clean architecture, avec un découpage en feature, et un dossier shared avec des services pour permettre le partage de certains éléments : appels API, widgets génériques, utilitaires (Dart permet les extensions de méthodes) et tout code abstrait méritant d'être appelé au nécessaire dans toutes les features. Attention à ne jamais négliger les injections de dépendances (préconisation de l'utilisation du package get_it). **Figure 1**

A noter *que dans le cas d'app cross-plaform souhaité, il peut être intéressant de jouer sur les injections de dépendances sur certaine couche (exemple : stockage dans le cache ou dans les cookies). La classe qui gère un storage est donc injectée selon la plateforme.*

// if Platform

```
final sl = GetIt.instance;

Future<void> init() async {
 // External
 final sharedPreferences = await SharedPreferences.getInstance();
 await Firebase.initializeApp();
 sl.registerLazySingleton(() => sharedPreferences);
 sl.registerLazySingleton(() => FlutterAppAuth());
 sl.registerLazySingleton(() => FirebaseMessaging.instance);
 sl.registerLazySingleton(() => const FlutterSecureStorage());
 sl.registerLazySingleton(() => Logger(
    level: const bool.fromEnvironment('dart.vm.product')
        ? Level.warning
        : Level.verbose));
 sl.registerLazySingleton(() => FirebaseAnalytics());

 // Core
 sl.registerLazySingleton<INetworkInfo>(() => NetworkInfo(sl()));
 if (Platform.isAndroid || Platform.isIOS)
  sl.registerLazySingleton(() => InternetConnectionChecker());
 sl.registerLazySingleton<ILoggerManager>(() => LoggerManager(sl()));

 //Services
 sl.registerLazySingleton<IAnalyticService>(() => AnalyticService(sl()));
 sl.registerLazySingleton<ICrashlyticsService>(() => CrashlyticsService());
 sl.registerLazySingleton<ILoggerService>(() => LoggerService(sl(), sl()));
}
```

Design pattern : BLOC

Chaque BLOC se compose d'un événement qui va ensuite impliquer un état. Selon l'état, on peut choisir ce que l'on souhaite voir s'afficher (un loader, le temps du chargement, un message d'erreur en cas de problème, une lecture selon les droits utilisateurs, etc…).

On peut retenir 4 états primaires les plus courants :

- Initial (je n'ai encore rien fait, je m'apprête à faire)
- Loading (j'ai commencé quelque chose, j'attends un résultat)
- Loaded (j'ai obtenu mon résultat, quel qu'il soit)
- Failure (j'ai obtenu un résultat qui n'était pas attendu ou quelque chose m'a empêché d'avoir un résultat...)

// event

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

abstract class ParametersEvent extends Equatable {
 @override
 List<Object> get props => [];
}

class GetParameters extends ParametersEvent {}

class GetLanguages extends ParametersEvent {}

class ModifyCurrentLanguage extends ParametersEvent {
 @override
 final int languageId;

 ModifyCurrentLanguage(this.languageId);

 @override
 List<Object> get props => [languageId];
}

class Disconnect extends ParametersEvent {}

class ModifyCurrentNotifications extends ParametersEvent {}

// state
abstract class ParamertersState extends Equatable {
 @override
 List<Object?> get props => [];
}

class ParametersInitial extends ParamertersState {}

class ParametersLoading extends ParamertersState {}

class ParametersLoaded extends ParamertersState {}
```

Exemple : je clique sur un bouton qui a pour but de m'afficher une nouvelle page et je provoque l'événement qui va initialiser les données dont j'ai besoin. Selon ce dernier, l'état peut être différent : trouvé, pas trouvé, en chargement, en erreur…

Widgets

C'est grâce à ces widgets que nos pages s'affichent sur l'application et que l'on peut donc produire des IHM.

- **Stateless**

C'est une page statique, le widget est sans état et ne bougera pas. Les vecteurs du moteur graphique n'auront pas à se redessiner sur cet élément. Il n'y aura donc pas d'interactions ce qui n'entraînera pas de modification de la page.

Ici un bouton, qui peut tout exécuter (Function() onPressed), avec un padding qui peut changer l'espace pour l'allonger/le rétrécir et qui prend un texte à afficher au milieu :

// Light blue button widget - stateless

```
class LightBlueButtonWidget extends StatelessWidget {
 final Function() onPressed;
 final String text;
 final EdgeInsetsGeometry? padding;

 const LightBlueButtonWidget(
    {Key? key, required this.onPressed, required this.text, this.padding})
    : super(key: key);

 @override
 Widget build(BuildContext context) {
  return TextButton(
    onPressed: onPressed,
    style: Theme.of(context).textButtonTheme.style!.copyWith(
        minimumSize: MaterialStateProperty.all(Size(50.w, 30.h)),
        padding: MaterialStateProperty.all(EdgeInsets.zero),
```

```
        enableFeedback: true,
        tapTargetSize: MaterialTapTargetSize.shrinkWrap,
        backgroundColor:
         MaterialStateProperty.all(Theme.of(context).primaryColorLight),
       ),
    child: Padding(
     padding: padding ?? EdgeInsets.symmetric(horizontal: 35.w),
     child: Text(
      text,
      style: Theme.of(context)
        .textTheme
        .button!
        .copyWith(fontWeight: FontWeight.normal, fontSize: 16.sp),
     ),
    ),
   );
 }
}
```

• **Statefull**

Ce widget a un état. Les vecteurs de l'IHM sont susceptibles de se redessiner. Par exemple, un bouton switch on/off. Au changement d'état, le vecteur va se redessiner.

//switch button statefull widget

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

class SwitchButtonWidget extends StatefulWidget {
 final bool switchValue;
 final Function()? function;
 const SwitchButtonWidget({Key? key, required this.switchValue, this.
function})
    : super(key: key);

 @override
 _SwitchButtonWidgetState createState() => _SwitchButtonWidgetState();
}

class _SwitchButtonWidgetState extends State<SwitchButtonWidget> {
 @override
 Widget build(BuildContext context) {
  return Switch(
   value: widget.switchValue,
   onChanged: (value) {
    if (widget.function != null) widget.function!();
   },
   activeColor: Theme.of(context).primaryColorLight,
   activeTrackColor: Theme.of(context).primaryColorLight.withOpacity(0.5),
   inactiveTrackColor: Theme.of(context).primaryColorLight.withOpacity(0.5),
  );
 }
}
```

Internationalisation

C'est important de le mettre en place dès le début même si on part sur une application seulement en français. Même si l'application n'est pas traduite de suite, intégrer l'internationalisation permet d'être prêt si l'application venait à évoluer. Cela va permettre l'usage des Locales (format de langue Flutter), par exemple pour afficher la date au bon format. Si le contexte de l'app le permet, l'internationalisation permet aussi d'afficher les dates et l'heure depuis le format UTC vers la time zone du système du device.

// Variable de traduction

// fr

```
 "unexpected_error": "Une erreur inattendue est survenue.",
 "cache_error": "Une erreur de cache est survenue.",
 "profil_hello_user": "Bonjour {userFirstName}",
 "profil_code": "code",
 "profil_privileges_title": "vos privilèges",
```

// en

```
 "unexpected_error": "Unexpected error detected",
 "cache_error": "Cache error detected",
 "profil_hello_user": "Hello {userFirstName}",
 "profil_code": "code",
 "profil_privileges_title": "your offers",
```

//japonais

```
"unexpected_error": "予期せぬエラーが発生しました",
"cache_error": "キャッシュエラーが発生しました",
"profil_hello_user": "こんにちは {userFirstName}",
"profil_code": "コード",
"profil_privileges_title": "あなたの特典",
```

```
if (state is ParametersLoaded)
 return Expanded(
  child: Text(S.of(context).profil_hello_user(state.firstName),
  style: Theme.of(context).textTheme.headline3,
  overflow: TextOverflow.ellipsis,
  maxLines: 1,
 ),
 );
```

Responsive

Pour une application responsive, il ne faut jamais développer avec des valeurs pixels en dur, mais des ratios de pixels (pourcentage). Des librairies sont disponibles, notamment le **Package screen_utils,** qui permet d'utiliser ces ratios de pourcentage.
Exemple : un container prend toujours 60% de l'écran quelle que soit la taille de l'écran (Idem pour la taille de police).

// size

```
   child: Padding(
    padding: padding ?? EdgeInsets.symmetric(horizontal: 35.w),
    child: Text(
     text,
     style: Theme.of(context)
       .textTheme
       .button!
       .copyWith(fontWeight: FontWeight.normal, fontSize: 16.sp),
    ),
   ),
  );
```

Penser responsive, penser accessibilité. Certaines personnes utilisent des add-ons pour agrandir la taille de police sur leur téléphone. Si la taille est en dur, l'add-on ne fonctionnera pas.

Best practices

- Définir son architecture et son découpage dès la création du projet permet d'assurer la maintenabilité de l'application (attention au copié-collé de tutoriel désorganisé)
- Appliquer les règles lint (Dart style guide) et les suivre au plus strict niveau pour augmenter la maintenabilité. Il ne faut jamais laisser un seul warning dans la console. Exemple : le mot clef « const » pour les widgets sera souvent demandé, car il permet d'alléger le travail du garbage collector (gestion des ressources utilisées par l'application : RAM et CPU).
- Utiliser les streams à bon escient. Présents dans tous les langages, ils peuvent causer les memory leaks et crash d'applications.
- Les Tests Driven Design (TDD) sont à appliquer autant que possible pour éviter les bugs et permettre un meilleur découpage des objets affichés dans l'IHM. Écrire ses tests unitaires avant d'écrire son code est une valeur ajoutée. Cela va réduire la quantité de bugs et améliorer la maintenabilité.
- Penser à documenter. Créer un fichier d'utilisation permet de recentrer les bonnes marches à suivre. Tout est réutilisable à condition de savoir comment l'utiliser (propriétés, ligne de commande…)
- Quand on écrit une IHM, il faut refactoriser ses widgets en permanence pour toujours avoir un minimum de code et que cela soit plus rapide pour se repérer. Il faut extraire ses widgets pour qu'ils deviennent génériques ou au moins réutilisables plusieurs fois au sein d'une même page
- Git flow : va avec les fichiers environnements. Il faut éviter de commiter ses modifications sur la branche main/master
- CI (Continues Intégration) / CD (Continues Deployment) va avec le Git Flow. Il faut, si possible, intégrer Firebase (outil gratuit géré par Google) pour pouvoir effectuer des tests sur un vrai device sans toucher à la prod. Aux vues du grand nombre de device, les tests réels permettent de déceler les éventuels morceaux de code qui ne fonctionnent pas et qu'il faut donc adapter pour une meilleure fluidité

Sécurité

- La sécurité des données est fondamentale. Les données sensibles doivent être stockées dans le secure storage (token, mots de passe…), car il est encrypté
- Brouiller la compilation en rajoutant un paramètre dans la ligne de commande appelée "obfuscate". Cela va permettre de rendre la décompilation très complexe pour empêcher la lecture et la récupération de données dans le code
- Détecter si le device a été rooté (Android) ou Jailbroken (iOS). Cela donne plus de droits et de privilèges, empêche l'exécution de code et permet de contrer les malwares. Si vous êtes également le développeur de l'API en plus de l'application : ajoutez un certificat SSL pour empêcher les attaques type Man In the Middle (MITM)
- Intégrer un système d'authentification type oAuth2, authentification à double facteur, empreinte digitale…
- Dans le cas de données sensibles, attention aux écrans capturés quand l'application est en tâche de fond (background snapshot). On peut empêcher l'affichage de la miniature d'écran via un package (https://pub.dev/packages/secure_application)

La solution Flutter a été pensée pour créer de manière simplifiée des applications. Pour les mettre en place, pensez à préparer sa création dès le début et respecter les best pratices que vous retrouvez dans la documentation officielle. Hiérarchisez l'arborescence, pensez à la sécurité de vos données et prévoyez des tests avant même de commencer à coder.

Liste des commandes essentielles Flutter

A noter que toutes ces commandes disposent d'options et que bon nombre d'entre elles sont déjà intégrées dans les IDE. Il n'est donc pas nécessaire de les lancer manuellement. Il existe également d'autres commandes.

flutter analyse	Permet l'analyse lint
flutter build	Permet de compiler le projet
flutter channel	Liste les canaux flutter disponibles. Cette commande permet de changer de canal
flutter config	Permet la configuration des paramètres de flutter
flutter create	Permet de créer un nouveau projet
flutter doctor	Liste les informations liées à l'installation de flutter (licences, problèmes éventuels, etc.)
flutter downgrade	Permet d'utiliser une version antérieure du canal courant de flutter
flutter run	Permet de lancer un programme flutter
flutter pub […]	Permet d'utiliser les packages pub
flutter test	Permet de lancer les tests
flutter upgrade	Permet de monter de version flutter

Déploiement de l'app sur Android et ios

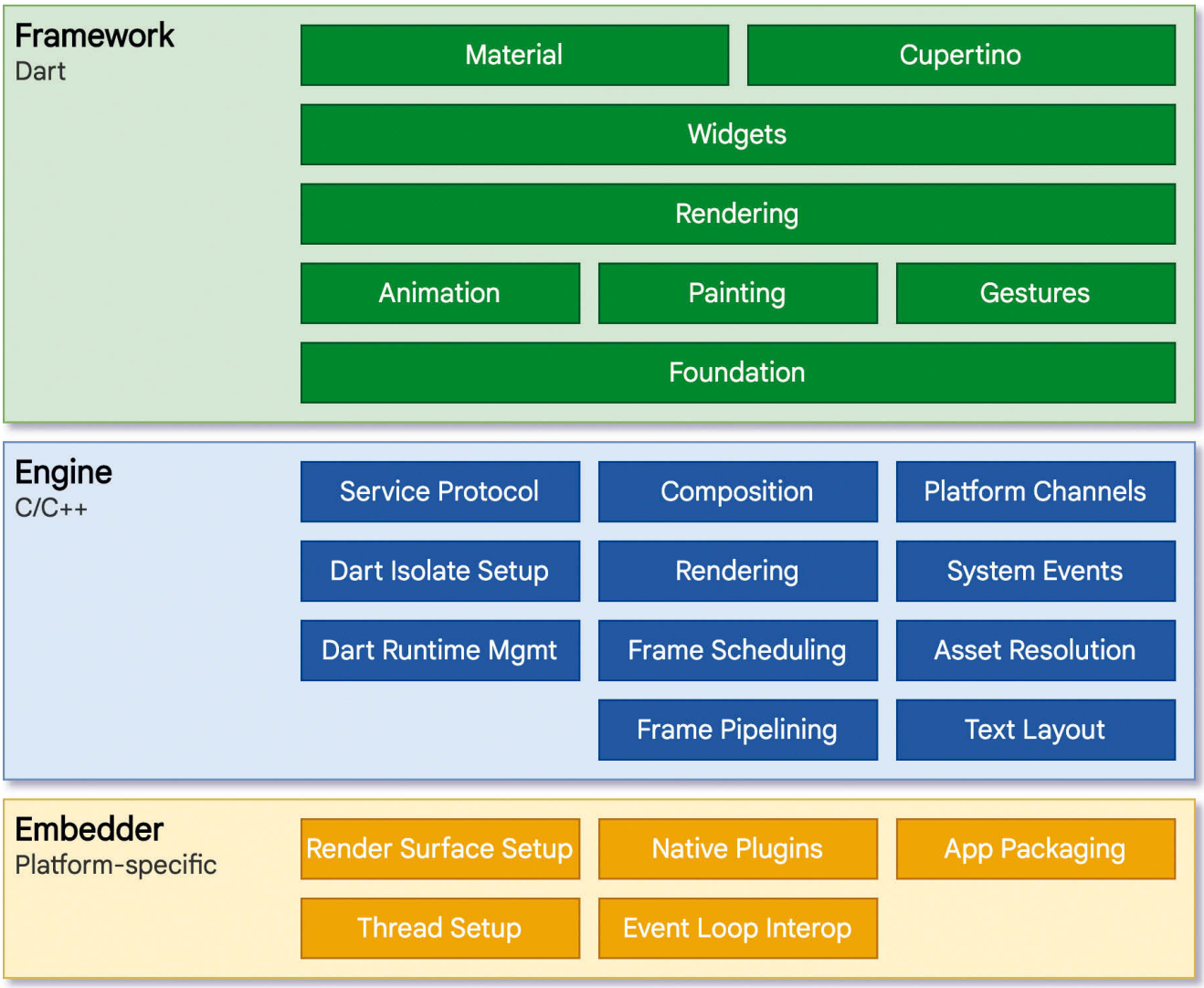
Il faut penser à passer par ces étapes en commun pour toutes les plateformes finales :

- Rajouter une icône de lancement (package flutter_launcher_icons)

Pour Android :

- Signer l'application (le fameux keystore pour Android)
- Configurer Gradle : pour cela, il faudra se référencer à la documentation officielle Android et/ou celle de Flutter
- **Note :** *builder avec la commande flutter « build appbundle ». Cette commande utilise le compilateur R8. Le code est buildé et automatiquement « shrink ». Par exemple, ce qui n'est pas utilisé est retiré ce qui allonge le temps de build mais permet une belle optimisation du package final. On parle de code shrinking, resource shrinking, obfuscation et optimization*
- **Note :** *si l'application est assez large, le multidex support sera demandé. Il faudra alors se référencer à la documentation officielle Android*
- Vérifier les AndroidManifest.xml (comme pour toutes app Android)
- Vérifier le fichier de build Gradle (comme pour toute app Android)
- Vérifier son versionning (dans le pubspec.yaml, il s'agit de la ligne « version : 1.0.0+1 » par défaut)
- Builder le bundle et non pas le apk (ce dernier est déprécié)
- Publier sur le Play Store via la Google Play Console

Architecture



Pour ios :

A noter que pour ios, cela requiert obligatoirement l'utilisation de macOS (utilisation de XCode).

- Préparer son application sur l'App Store Connect en suivant les recommandations d'Apple : créer un compte développeur, créer le Bundle ID unique à l'application, renseigner les informations de son application…
- Vérifier les paramètres de son projet via XCode (renseigner notamment son Bundle ID)
- Vérifier son AppframeworkInfo.plist notamment pour la version minimale de déploiement «MinimumOSVersion »
- Signer l'application (les certificats Apple et provisionning profile)
- Publier son appli sur TestFlight

Dans les 2 cas, il faut toujours penser à vérifier la documentation officielle de Flutter (https://docs.flutter.dev/, menu « Deployment »). Elle reflètera toujours les nouveautés et pré requis pour Google et Apple, et donc précisément ce que vous devez faire pour être dans les normes et ne pas voir votre application être rejetée.

A noter qu'il faut bien compter 3 semaines entre le moment où l'on souhaite publier son appli et le fait de voir son appli sur un store.

Conclusion

En plus de deux ans, Flutter est devenu un incontournable. Même si le parfait n'existe pas, la technologie Flutter est irréprochable en termes d'efficacité (optimisation, affichage FPS très élevé) quand on la compare aux autres cross platform. L'écriture du code des IHM est si simple que l'on peut presque dire qu'il n'est pas possible de faire des affichages qui ne sont pas beaux.

La technologie Flutter permet donc d'avoir, en peu de temps, une application jolie et efficace. Il est possible d'écrire son propre code Swift/Android et de le relier via les platform channels.

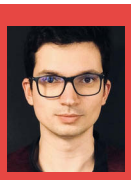
Néanmoins, concernant Skia, le moteur graphique utilisé par Flutter, certains éléments natifs ne l'utilisent pas les rendant parfois plus compliqués à utiliser et plus lents (webview, les cartes GPS…).

A noter qu'il n'est pas possible d'utiliser Flutter pour tvOS, watchOS, CarPlay et Android Auto.

Enfin, dans certains cas, il faudra privilégier un autre langage cross platform ou bien un code natif, avec l'écriture de deux applications distinctes (Android et ios), surtout si l'on cherche une application extrêmement réactive. Un cross platform est forcément plus lent.

JavaScript : quel framework choisir ?

Aujourd'hui dans l'univers des frameworks JS, nous avons tous déjà entendu parler d'Angular, de React ou de Vue, etc. Ils sont très populaires et répondent à de nombreux usages. Très rapidement la question se pose, comment choisir ? Dans ce dossier, nous allons voir comment faire un choix entre ces frameworks.



Jimmy Kasprzak

Développeur formateur et animateur de communauté chez Zenika

Passionné par le domaine du web et l'écosystème JS.

Éleveur de pokémons et combattant de la faille de l'invocateur à ses heures perdues

Twitter: @JimmyKasprzak

zenika

Je ne parlerai ici que d'Angular, React et Vue. Deux raisons : la première est que je considère ces trois frameworks JS front-end comme ceux étant les plus utilisés à l'heure actuelle. La deuxième raison est que je n'ai d'expériences significatives que sur ces trois frameworks. Cela ne veut pas dire qu'il n'existe aucun autre framework tout aussi pertinent et fonctionnel pour un projet.

Quels critères ?

La première question à se poser est : quels critères dois-je prendre en compte pour mon choix ? Une multitude de critères peuvent être considérés et il est très facile de se sentir dépassé ou perdu. Je vais présenter ici les critères que je considère comme étant parmi les plus pertinents dans le choix d'un framework.

Avant de rentrer dans les détails de ces critères, une petite précision : je tente de fournir ici des analyses les plus objectives possibles. Cependant, il faut avoir conscience que l'importance accordée à ce qui ressort de ces analyses est subjective. Chacun d'entre nous accordera par exemple plus ou moins d'importance à la présence d'une documentation claire et fournie.

Documentation

Le premier critère qu'on peut utiliser pour comparer Angular, React et Vue est la documentation. À ce niveau, aucun des trois frameworks ne présente un retard conséquent vis-à-vis d'un autre. Les trois disposent d'une documentation officielle fournie, avec des exemples, une documentation de référence d'API, des guides avancées, etc.

Un petit avantage cependant pour Angular et Vue, qui présentent une documentation un petit peu plus complète et variée que celle de React. Il est important de noter qu'Angular se veut un framework complet, avec une solution officielle pour la majorité des scénarios d'utilisation d'une web app. Il paraît donc logique d'y trouver un peu plus de documentation. Dans tous les cas, la documentation officielle est suffisamment fournie pour ne pas être un point bloquant dans le choix d'un de ces frameworks.

Communauté

On ne peut parler de documentation sans évoquer la communauté autour d'un framework. En effet, celle-ci est la source de nombreuses ressources en ligne, que ce soit au niveau de la documentation officielle ou non. C'est donc également un critère de comparaison important.

Quand on parle de communauté autour d'un framework, la question porte principalement sur le nombre de développeurs adeptes du framework (de manière professionnelle ou non).

Il est très difficile d'estimer le nombre de personnes utilisant un certain framework. Beaucoup de références différentes existent. On trouve par exemple le rapport annuel de "stateofjs", le nombre de stars sur github, le nombre de téléchargements depuis npm ou encore le nombre de visites sur le site de la documentation officielle. En fonction de la référence utilisée, la popularité - et donc la communauté - de tel ou tel framework est plus ou moins importante.

Toujours est-il qu'aujourd'hui les trois frameworks disposent chacun d'une communauté très large. Y a-t-il quelques milliers de développeurs de plus sur un framework plutôt qu'un autre ? Oui sans aucun doute. Est-ce que cela fait une réelle différence ? Je ne pense pas.

Que ce soit Angular, React ou Vue, les trois frameworks disposent d'un nombre de ressources très conséquent - en plus de la documentation officielle. Au point que lorsqu'on rencontre un problème pour réaliser quelque chose avec le framework, il est presque certain qu'il existe déjà une ressource en ligne pour nous aider.

Donc oui, il est sans aucun doute vrai qu'il existe de petites différences de popularité entre les trois frameworks. Mais chacun dispose de toute manière d'une communauté massive, ce qui fait que la question "Quel framework dispose aujourd'hui de la plus grande communauté ?" ne présente plus vraiment d'importance.

Performance

Manipulation du DOM (ms)

Framework	create rows	replace all rows	partial update
Angular (v12.0.1)	133.4	110.1	181.6
React (v17.0.1)	136.1	112.8	218.1
Vue (vue-v3.2.1)	107.7	98.7	203.3

Startup (ms)

Framework	consistently interactive
Angular (v12.0.1)	2 444.7
React (v17.0.1)	2 581.4
Vue (vue-v3.2.1)	2 105.1

source: https://rawgit.com/krausest/js-framework-benchmark/master/webdriver-ts-results/table.html

startup (ms)

Une question qui revient très souvent lorsqu'on cherche à déterminer quel framework utilisé est la performance. Pour comparer les trois frameworks, une source couramment

utilisée est la performance benchmark réalisée par Stefan Krause (https://krausest.github.io/js-framework-benchmark/2021/table_chrome_93.0.4577.63.html).

Lorsqu'on parle de performance d'un framework JS, on cherche à regarder essentiellement deux choses. La première est le temps que va mettre notre application à démarrer (avoir une première "page" interactive). La deuxième est le temps nécessaire à la manipulation du DOM de notre application.

Je ne montre ici qu'une partie des analyses réalisées dans ce benchmark, n'hésitez pas à aller voir par vous-même l'intégralité des données recueillies. Celles-ci se trouvent facilement en ligne.

En ce qui concerne le temps nécessaire à l'obtention d'une page interactive, il apparaît que la dernière version de Vue est un peu plus performante que Angular et React.

Au niveau de la manipulation du DOM, là aussi on peut constater ici qu'il y a une légère différence entre les frameworks et que globalement Vue a l'air de s'en sortir un peu mieux que ses compères.

Au final, Vue semble tirer son épingle du jeu au niveau des performances. Cependant, les chiffres restent malgré tout dans la même gamme, au point que la question des performances ne permet pas, à mon avis, de faire une distinction significative entre les frameworks.

De plus, l'expérience que j'ai acquise sur ce genre de problématique me laisse penser que les principaux problèmes de performance qu'on va rencontrer dans une application web sont rarement dus au framework choisi (je ne parle ici que du cas où le framework est au choix Angular, React ou Vue). Ils sont souvent plutôt dus à un besoin fonctionnel mal exprimé ou une architecture technique défaillante - qu'on essaye de compenser côté front. Par exemple, si on a besoin d'afficher au même moment 20 000 éléments différents dans notre application, cela prendra du temps, peu importe le framework employé. La question qui se pose dans ce cas est plutôt : quelle est l'utilité d'afficher autant d'éléments en une fois ?

Langage

Angular, depuis sa sortie officielle en 2016, a toujours préconisé le langage Typescript (je ne considère pas ici Angular.js (version 1.x) qui même s'il partage un nom, est un framework complètement différent d'Angular (version 2+)). Au point qu'aujourd'hui utiliser Angular dans un projet professionnel sans Typescript est définitivement une mauvaise idée. En effet, la majorité des ressources en ligne et la documentation officielle ne traite d'Angular qu'en Typescript. Par ailleurs, le framework lui-même est écrit en Typescript, ce qui assure un très bon niveau d'intégration avec ce langage.

Vue (dans sa version 3) a lui aussi été rédigé en Typescript et promeut l'utilisation de ce langage. Enfin, bien que React n'ait pas été rédigé en Typescript, il permet lui aussi à ses développeurs de l'utiliser.

Nous pouvons donc constater que dans les 3 cas, il est possible d'utiliser Typescript. Quelle proportion des projets réalisés avec ces frameworks est écrite en Typescript ? Il est impossible de le dire. On peut cependant affirmer avec certitude que dans le cas d'Angular, la très grande majorité des projets s'en servent. Pour Vue et React, la difficulté à utiliser Typescript ne vient pas tellement des frameworks, mais de leur écosystème. Étant principalement disponible en JS à l'origine, Typescript est moins couramment utilisé qu'avec Angular. De ce fait, les librairies disponibles (spécifiques à ces frameworks) ont tendance à être moins souvent adaptées à une utilisation avec Typescript. Il faut cependant noter que cette difficulté est de moins en moins présente et qu'il est de toute façon tout à fait possible d'utiliser une librairie purement JS (sans définition de types par exemple) dans un projet Typescript. Simplement dans ce cas, on ne disposera pas de toutes les fonctionnalités offertes par Typescript.

Cependant, à la différence d'Angular, il existe un nombre conséquent de ressources qui documentent l'utilisation de Vue et React avec du JavaScript. Ce qui fait qu'il est tout à fait possible et pertinent de préférer utiliser JavaScript avec ces derniers.

Au final, si vous souhaitez utiliser Typescript alors les trois frameworks vous seront accessibles. Si cependant vous ne souhaitez utiliser que du JS, alors il faudra plutôt se tourner vers React ou Vue.

Licence

Angular, React et Vue sont tous les trois publiés sous une licence MIT. Aucun souci particulier à se faire quant à leur utilisation, que ce soit dans le cadre d'un projet personnel ou professionnel, sources de revenu ou pas.

Maintenance et évolution

Chacun de ces frameworks a une équipe dédiée à sa maintenance et son évolution. Le travail de ces équipes est directement visible sur GitHub, car Angular, React et Vue y possèdent chacun un dépôt avec leur code source.

Angular a été créé et est maintenu par une équipe de développeurs de Google. React suit un modèle similaire, mais provient de chez Facebook. Vue se différencie de ses compères, car aucune grande entreprise ne se trouve directement derrière son maintien et son évolution. C'est Evan You qui est à l'origine de ce framework.

Il faut cependant bien comprendre qu'il existe malgré tout pour Vue, comme avec Angular et React, une "core team" en charge de son développement.

De plus, que ce soit pour Angular, React ou Vue, les contributions de la communauté sont non seulement acceptées, mais aussi encouragées. Il existe d'ailleurs pour chacun d'entre eux un guide de contribution consultable sur leur dépôt GitHub respectif.

En ce qui concerne l'évolution de ces frameworks, ils fonctionnent de manière similaire là aussi.

Angular suit la norme du semantic versioning: major.minor.patch.

Une nouvelle version majeure sort en moyenne tous les 6 mois. Cela peut sembler beaucoup, mais dans la réalité ces nouvelles versions majeures introduisent assez peu de changements nécessitant une action d'un développeur (cela dépend évidemment de la taille de votre application). Ce rythme de release a avant tout pour objectif de garder un rythme stable sur l'évolution du framework.

Pour ce qui est des LTS, Angular supporte chaque version majeure sortie pour au minimum 18 mois. En échange, un réel effort est fait pour aider à la migration et s'assurer que nos projets restent à jour le plus facilement possible. Un exemple est l'existence de "update.angular.io", une application officielle nous indiquant les étapes à suivre pour passer d'une version à une autre.

React dispose lui aussi d'une page de documentation officielle dédiée à sa politique de gestion des versions. Là aussi on trouve du semantic versioning. React a introduit très peu de breaking changes au fur et à mesure du temps et n'a créé que très peu de nouvelles versions majeures ces derniers temps. React 17 est sortie en mars 2020, la version 16 en septembre 2017. Les nouvelles fonctionnalités sont surtout ajoutées avec les nouvelles versions mineures.

Enfin, Vue utilise lui aussi le semantic versioning et sort très peu de nouvelles versions majeures. Comme les deux autres frameworks, le détail de chaque nouvelle release (que ce soit une nouvelle version de patch, mineure ou majeure) est indiqué. Pour chaque nouvelle version majeure, la précédente version mineure sera maintenue 18 mois.

Support des navigateurs

En ce qui concerne le support des navigateurs, il n'existe pas non plus beaucoup de différences entre Angular, React et Vue. Les trois frameworks supportent les dernières versions de tous les navigateurs modernes (Chrome, Firefox, Opéra, Edge, Safari).

- Angular vient d'arrêter définitivement le support de IE avec sa dernière version majeure (13 à l'heure où est écrit ce dossier). Avant cela, IE 11 était encore supporté à l'aide d'un fichier de polyfills à rajouter dans son application.
- React supporte jusqu'à IE 8 à l'aide d'un polyfill également.
- Enfin, Vue se décrit comme supportant tous les navigateurs embarquant un moteur JavaScript compatible ES5, autrement dit supportant jusqu'à IE 9.

Techniquement la version la plus "ancienne" supportée par ces frameworks diffèrent donc un peu. Cependant, il est important de préciser que IE est aujourd'hui complètement déprécié et ne reçoit plus aucune mise à jour. Décider de supporter ce navigateur, c'est allé au-devant de problèmes qui ne feront que s'aggraver au fur et à mesure du temps, car il est très probable que React et Vue - ainsi que tous les autres frameworks JS - abandonnent complètement le support de IE dans leurs prochaines versions.

Maturité et stabilité

Les trois frameworks existent depuis déjà "longtemps" - longtemps étant relatif à la vitesse de changement existante dans le monde JS. Dans l'ordre, la première release de React date de mai 2013, celle de Vue date février 2014 et celle d'Angular de septembre 2016.

Il faut également préciser que pour le développement d'Angular, les équipes de Google ont pu s'appuyer sur les années d'expérience avec Angular.js.

Comme évoqué un peu plus tôt dans ce dossier, les trois sont maintenus et reçoivent régulièrement des mises à jour, que ce soit pour du bug fix, de l'optimisation ou de l'ajout de fonctionnalités.

Les nouvelles versions publiées de chacun de ces frameworks sont stables. Il arrive évidemment que pour passer d'une nouvelle version à une autre, des changements soient nécessaires dans le code de notre projet. Mais la majorité du temps, ces changements s'effectuent sans difficulté majeures (je ne parle pas ici des éventuelles librairies non officielles importées dans un projet, mais bien uniquement du framework directement).

Un nombre conséquent de projets a déjà été réalisé avec ces frameworks. Il est difficile de citer des chiffres précis étant donné que beaucoup de projets d'entreprises ne sont pas accessibles publiquement. On peut cependant le déduire en se basant sur nos propres expériences personnelles et professionnelles, sur la communauté importante et active dont dispose chacun des frameworks et sur le nombre important de ressources en ligne.

Là encore, il n'y a donc pas de réelles distinctions entre les trois frameworks. Les trois sont considérés comme matures et stables.

Petite précision avec Vue. Il faut savoir que le framework a publié le 18 septembre 2020 sa dernière version majeure : Vue 3. Bien que cette version soit stable, l'écosystème autour de Vue est encore dans une période d'adaptation. Lors du passage de Vue 2 à Vue 3, la plus grande difficulté qu'on peut rencontrer consiste à s'assurer que toutes les dépendances de notre projet soient bien compatibles avec Vue 3.

Enfin, basées sur mon expérience, les migrations de versions d'Angular ont toujours été plus simples à réaliser qu'avec React. Cela s'explique selon moi par une des différences fondamentales entre Angular et React: Angular se définit comme un framework complet, offrant une solution officielle pour gérer tous les aspects d'une application web. React est beaucoup plus minimaliste. Même si à première vue cela semble indiquer que React sera plus simple à monter en version que React, c'est en réalité plutôt l'inverse.

Dans un projet réel React, on a tendance à se retrouver avec beaucoup plus de librairies tierces produites par la communauté que dans un projet Angular. Ce qui fait d'autant plus de sources potentielles de problèmes au moment de monter en version, car il va falloir s'assurer que chacune de ces librairies tierces soit compatible.

À l'inverse, lorsque Angular monte en version, cela signifie que tous ses packages officiels sont prêts à fonctionner avec cette version.

Ce n'est évidemment que mon expérience, et cela ne veut pas dire pour autant que chaque montée de version de React se fait forcément dans la douleur !

Fonctionnalités

Un des critères de sélection les plus importants dans le choix d'un framework est bien évidemment les fonctionnalités offertes.

Plutôt que de faire une liste à n'en plus finir de toutes les fonctionnalités qu'offrent chacun Angular, React et Vue, je vais me concentrer sur quelques fonctionnalités qu'on retrouve, à mon sens, dans presque toute application web :

- Afficher des données à l'écran et de réagir à des interactions de l'utilisateur
- Routing

- Un mécanisme de partage de code entre les différentes parties de notre application
- Gérer des formulaires
- Déclencher des requêtes http

Composants

Angular

Un des concepts au cœur même aussi bien d'Angular, React ou Vue est la programmation orientée composant. Un composant se définit par une logique (du code) associée à une vue (un template). En utilisant ce concept avec un mécanisme de composition qui nous permet d'utiliser des composants dans d'autres composants, on finit ainsi par créer une application complète.
Un composant Angular se définit à l'aide d'une classe Typescript et d'un template HTML.

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  data: Product;
  isNextButtonDisabled = false;

  goToNextProduct(): void {
    // Go to next product
  }
}
```

```
<section>
  <h1>{{ data.title }}</h1>
  <span>{{ data.price }}</span>

  <button
    [disabled]="isNextButtonDisabled"
    (click)="goToNextProduct()"
  >
  Next
  </button>
</section>
```

La liaison entre la class typescript et le template HTML se fait à travers de métadonnées dans l'annotation @Component.
Les vues des composants Angular sont définies avec un langage de templating qui ressemble au HTML, avec des fonctionnalités supplémentaires. Dans cet exemple:

- Les {{ }} indiquent l'utilisation d'interpolations. Ce mécanisme permet d'indiquer une expression JavaScript (à quelques exceptions près) et de voir cette expression remplacée par sa valeur à l'écran.
- Les [] indiquent du property binding. Cela permet de lier une propriété du DOM à un variable.
- Les () s'utilisent pour se mettre en écoute sur un évènement. Cela permet notamment de détecter des interactions de l'utilisateur dans notre application et d'y réagir en conséquence.

En plus de cela, il me paraît important d'illustrer comment deux composants communiquent entre eux en utilisant le concept de composition.

```
<section>
  <app-product></app-product>

  <button
    [disabled]="isNextButtonDisabled"
    (click)="nextProduct()"
  >
  Next
  </button>
</section>
```

On remarque que la balise "app-product" est particulière. Elle ne correspond à aucun élément HTML natif. Cet élément va servir à Angular pour savoir quel composant créer et afficher. C'est ainsi que s'utilise le principe de composition.
Très souvent il va être nécessaire de faire transiter des informations d'un composant à un autre. En Angular, une manière de faire est d'utiliser le property binding et l'event binding, exactement comme on l'a vu dans l'exemple précédent sur l'élément HTML "button".

```
<section>
  <app-product
    [product]="data"
    (addToBasket)="addProduct">
  </app-product>

  <button
    [disabled]="isNextButtonDisabled"
    (click)="nextProduct()"
  >
  Next
  </button>
</section>
```

Ici, nous fournissons au composant représenté par la balise *app-product* une propriété "product" avec pour valeur ce que contient la variable "data".
Nous écoutons également l'évènement « addToBasket » qui sera émis depuis ce même composant.

```
@Component({
  selector: 'app-product',
  templateUrl: './app.product.component.html',
  styleUrls: ['./app-product.component.css']
})
export class ProductComponent {
  @Input() product: Product;
  @Output() addToBasket = new EventEmitter<Product>();

  add(): void {
    this.addToBasket.emit(this.product);
  }
}
```

En ajoutant une annotation @Input sur une propriété de notre composant, on indique que cette propriété peut être modifiée depuis l'extérieur de ce composant. Autrement dit, on donne la possibilité à un composant parent (un

composant utilisant le ProductComponent dans son template) de définir la valeur de la propriété "product". On établit donc une liaison dans un sens : du composant parent vers le composant enfant.
En ajoutant une annotation @Output sur un objet de type EventEmitter, on permet à ce composant d'émettre l'évènement du même nom vers l'extérieur. C'est très utilisé pour faire remonter des informations sur quelque chose qui vient de se produire dans un composant enfant, notamment parce qu'il est possible de transporter des données dans cet évènement. C'est ce qu'on peut voir dans la méthode "add". Nous utilisons l'EventEmitter et sa méthode "emit" pour émettre notre évènement avec des données, ici le contenu de la variable "this.product".
On établit ainsi une liaison dans l'autre sens : du composant enfant vers le composant parent.
Après avoir compris cela, on peut se demander comment communiquer entre composants lorsque leur relation n'est pas aussi simple qu'un parent et un enfant. Nous en discuterons un peu plus tard dans ce dossier.
Pour React et Vue, les exemples seront en JS

React

Il existe deux manières de créer des composants avec React: avec une classe ou une fonction. Je ne détaillerai ici que la façon de faire avec une fonction, étant la manière de faire la plus populaire et la plus recommandée aujourd'hui par React.

```
function App(props) {
  const [data, setData] = useProduct({ title: 't-shirt', price: 10 });
  const nextProduct = () => {
    setData({ title: 'hat', price: 8 });
  }

  return (
    <section>
      <h1>{ data.title }</h1>
      <span>{ data.price }</span>

     <button disabled={isNextButtonDisabled} onClick={nextProduct}>
        Next
      </button>
    </section>
  )
}
```

Un composant React commence toujours par une majuscule. En React le nom de la fonction correspond au nom du composant.
Au sein d'un composant créé avec une fonction, la logique est représentée par le corps de la fonction. La vue est représentée par le retour de la fonction.
Pour décrire une vue, React met à disposition une syntaxe particulière appelée JSX. Cette syntaxe est transformée en fonction JS à terme. L'objectif de cette syntaxe est de tirer parti de notre connaissance du JavaScript et de l'appliquer à la description de notre vue.
La gestion des évènements en React est très proche de celle du JS natif. Au lieu d'utiliser des noms d'événements en minuscule comme "onclick", ils sont nommés en kebabCase "onClick". C'est ce que l'on voit dans l'exemple au niveau du bouton. Chaque clique sur celui-ci déclenche la fonction "nextProduct". Cette fonction va modifier le "state" (l'état) de notre composant.
Deux choses vont piloter un composant React, son "state" et ses "props". Le state représente l'état du composant. La variable "data" de notre exemple fait partie du state de notre composant, car créée à l'aide de la fonction "useState" (fournie par React). Si le state est modifié, React le détecte automatiquement et met à jour notre vue. Le deuxième élément est les props. Ce sont les inputs de notre composant et sont reçus en paramètre de notre fonction composant. Même chose, si une valeur des props change, notre vue est mise à jour.
Ces inputs permettent d'établir une communication bidirectionnelle. En effet, dans ces inputs nous pouvons envoyer des données, mais aussi des fonctions. Charge alors au composant qui reçoit ces fonctions de les appeler quand bon lui semblera.

```
function App(props) {
  const [data, setData] = useProduct({ title: 't-shirt', price: 10 });
  const nextProduct = () => {
    setData({ title: 'hat', price: 8 });
  }

  return (
    <section>
      <Product product={data} onClick={() => console.log('add product')} />

      <button disabled={isNextButtonDisabled} onClick={nextProduct}>
        Next
      </button>
    </section>
  )
}
```

```
function Product(props) {
  return (
    <section>
      <h1>{props.product.title}</h1>
      <span>{props.product.price}</span>

      <button onClick={props.onClick}>Ajouter au panier</button>
    </section>
  )
}
```

Ici, lorsqu'un utilisateur clique sur le bouton de notre composant Product, cela déclenche la fonction onClick reçue dans ses props. Cette fonction est fournie par le composant App.

Vue

Pour Vue, la manière recommandée de créer un composant est d'utiliser un format appelé SFC (Single File Component). L'idée étant, comme en React, de trouver tous les éléments de notre composant dans le même fichier. Ces fichiers possèdent une extension .vue et sont compilés par un moteur

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

PROGRAMMEZ!

Le magazine des dévs

NOS CLASSIQUES

1 an → 10 numéros (6 numéros + 4 hors séries) — 55€ *(1)

2 ans → 20 numéros (12 numéros + 8 hors séries) — 90€ *(1)

Etudiant
1 an → 10 numéros (6 numéros + 4 hors séries) — 45€ *

Option : accès aux archives — 20€

** Tarifs France métropolitaine*

(1) Au lieu de 69,90 € ou 139,80 € selon l'abonnement, par rapport au prix facial.

ABONNEMENT NUMÉRIQUE

PDF ..... 45€
1 an

Souscription directement sur
www.programmez.com

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

ABONNEMENT à retourner avec votre règlement à :
PROGRAMMEZ, Service Abonnements
57 Rue de Gisors, 95300 Pontoise

☐ Abonnement 1 an : 55 €

☐ Abonnement 2 ans : 90 €

☐ Abonnement 1 an Etudiant : 45 €
Photocopie de la carte d'étudiant à joindre

☐ Option : accès aux archives 20 €

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

Adresse email indispensable pour la gestion de votre abonnement

E-mail : @

☐ Je joins mon règlement par chèque à l'ordre de Programmez !

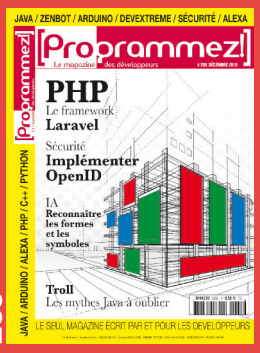
☐ Je souhaite régler à réception de facture

* Tarifs France métropolitaine

Offres pouvant s'arrêter à tout moment, sans préavis **PROG 251**

Les anciens numéros de PROGRAMMEZ!

Le magazine des développeurs



235

236

239

240

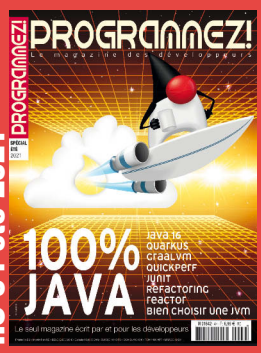
241

HS 01 été 2020

242

246

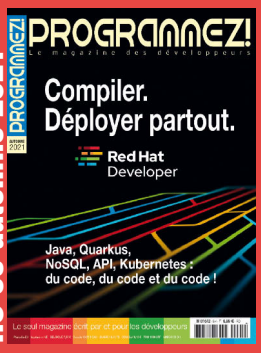
247



HS 04 été 2021

249

250



HS 05 automne 2021

Complétez votre collection....

Tarif unitaire 6,5 € (frais postaux inclus)

☐ *235*	: __ ex	☐ *241*	: __ ex	☐ *247*	: __ ex
☐ *236*	: __ ex	☐ *HS1 été 2020*	: __ ex	☐ *HS4 été 2021*	: __ ex
☐ *239*	: __ ex	☐ *242*	: __ ex	☐ *249*	: __ ex
☐ *240*	: __ ex	☐ *246*	: __ ex	☐ *250*	: __ ex
				☐ *HS5 automne 2021*	: __ ex

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 251**

du framework en fichier standards JS et CSS.
Les exemples sont rédigés en Vue 3.

```
<template>
  <section>
    <h1>{{ data.title }}</h1>
    <span>{{ data.price }}</span>

    <button :disabled="isNextButtonDisabled" @click="nextProduct">
      Next
    </button>
  </section>
</template>

<script>
import { defineComponent } from 'vue';

export default defineComponent({
  name: 'App',
  setup(props, context) {
    const isNextButtonDisabled = ref(false);
    const data = ref({ title: 't-shirt', price: 10 });

    const nextProduct = () => {
      data.value = { title: 'hat', price: 8 };
    };

    return { isNextButtonDisabled, data, nextProduct }
  }
})
</script>
```

On trouve au sein de la balise "template" toute la partie vue de notre composant. Comme avec Angular, un système d'interpolation à double accolage permet d'évaluer une expression et d'afficher sa valeur.
La logique de notre composant se trouve entre les balises "script", sous la forme d'un objet. Au sein de cet objet se trouve une propriété "setup" au sein de laquelle nous définissons les données et les traitements disponibles dans notre composant.
Tout comme dans React et Angular, nous retrouvons la notion de props et de state. Les props sont fournis par un composant parent et provoquent un rafraîchissement de la vue du composant si elles sont modifiées.
L'état d'un composant est représenté par des variables créées à l'aide de la fonction "ref" (la fonction "reactive le permet aussi). En passant par cette fonction pour créer nos variables, Vue va être capable de détecter les changements qui surviennent dans nos variables et mettre à jour la vue de notre composant en conséquence.
On peut affecter les propriétés d'un élément de notre template en utilisant le nom de la propriété préfixée d'un ":". Quant au mécanisme d'écoute d'un événement, il se base sur le nom de celui-ci suffixé d'un "@".
Ces deux mécanismes permettent également de communiquer d'un composant parent à un composant enfant et inversement, de manière très similaire à ce qu'on trouve en Angular. Il faudra pour cela indiquer explicitement les propriétés attendues dans une propriété du nom de "props" du composant enfant. Et pour être capable d'émettre un évènement custom, celui-ci devra être déclaré de la même manière dans une propriété "emits" (ce n'est pas obligatoire, mais vivement recommandé pour éviter des bugs).

```
<script>
  export default defineComponent({
    props: {
      product: Object
    },
    emits: ['addToBasket']
  })
</script>
```

Tout ça pour quoi ?
Le concept de composant est omniprésent au sein d'Angular, React et Vue. Chacun d'entre eux l'implémente de manière légèrement différente, mais le concept reste le même: une vue et de la logique reliées entre elles pour pouvoir interagir facilement. Une application construite avec l'un de ces frameworks consiste en une multitude de composants interagissant entre eux.

Routing

Quand je parle de routing ici, j'entends uniquement la mise en place côté front d'une gestion des changements dans l'URL.
Les trois frameworks partagent là aussi un concept commun : chaque URL de notre application va correspondre à un composant. C'est une navigation orientée composant.
Pour Angular, il existe un package officiel pour cela : **@angular/router**. Ce package nous donne accès à énormément d'éléments pour nous faciliter la gestion d'un routing côté front.

```
const routes: Routes = [
  { path: 'basket', component: BasketComponent },
  { path: '', component: HomeComponent }
];

@NgModules({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

La définition de routes dans notre application se fait à travers une liste d'objets. Dans un de ces objets, on trouve un "path". Celui-ci est relatif à l'URL de base de notre application (localhost:4200 en phase de développement par exemple). On trouve aussi une propriété "component". Celle-ci permet de désigner quel composant doit être créé lorsque le path associé est détecté dans l'URL.
Ici on indique donc qu'à la racine de notre application nous affichons le composant "HomeComponent" et que sur le path "basket" (localhost:4200/basket par exemple), nous affichons cette fois le composant "BasketComponent".
Le routeur d'Angular offre énormément d'autres fonctionnalités. Gestion de paramètres dans l'URL, gestion de routes imbriquées, mise en place de gardes, lazy-loading, etc. Il se veut être un package complet fournissant tous les

outils nécessaires à la mise en place d'un routing.
On peut noter également que la configuration des routes dans une application Angular se fait de manière statique. Le chargement de notre configuration est fait à l'initialisation de notre application. Il est cependant possible de modifier cette configuration même une fois l'application chargée.

React ne dispose pas de mécanismes de routage par défaut. Rien ne nous empêche de mettre en place une gestion de routes en utilisant du JS natif. Cependant, dans la réalité, un package de la communauté est utilisé presque à chaque fois pour cela: **react-router** et notamment son implémentation pour le web: **react-router-dom**.

```
export default function App() {
  return (
    <Router>
      <Route path="/basket">
        <Basket />
      </Route>
      <Route path="/">
        <Home />
      </Route>
    </Router>
  )
}
```

Avec ce package, la gestion des routes avec React est similaire à celle d'Angular dans le sens où on associe une URL à un composant. La différence se situe dans le fait qu'ici la définition de nos routes est dynamique. C'est le fait de créer le composant App qui permet à notre configuration d'être chargée. Avec Angular, les routes sont chargées à l'initialisation de notre application.
La majeure différence entre une configuration statique et une configuration dynamique repose dans la gestion de scénarios un peu plus complexes. L'idée de la configuration React est de pouvoir gérer ces scénarios de la même manière qu'on va gérer l'affichage d'éléments HTML classique. On va pouvoir par exemple supprimer une route en fonction d'une condition, de la même manière qu'on supprimerait un élément HTML. Quoi qu'il en soit, il est tout à fait possible d'obtenir des fonctionnalités similaires à celles offertes par défaut en Angular, il faudra par contre passer par une librairie tierce.

Vue possède un package reconnu officiellement pour mettre en place du routing dans une application: **vue-router**. La configuration des routes avec vue-router ressemble plus à celle d'Angular que React dans le sens où elle est statique.

```
const routes = [
  { path: '/', component: Home },
  { path: '/basket', component: Basket },
];

const router = VueRouter.createRouter({
  history: VueRouter.createWebHashHistory(),
  routes
});

const app = Vue.createApp({});
app.use(router);
```

De la même manière qu'Angular, il est possible de modifier dynamiquement cette configuration si nécessaire. De plus, ce package offre globalement les mêmes possibilités qu'avec @angular/router ou react-router.
Que peut-on conclure de tout cela sur la gestion du routing avec Angular, React et Vue ?
Qu'ils sont très similaires sur ce point également. Le concept est identique au travers des trois frameworks: on associe un composant à une URL. Il est évidemment possible de pousser les choses bien plus loin que les exemples présentés, mais les fonctionnalités plus avancées sont similaires aussi.
Même le fait de disposer d'un routage par défaut statique ou dynamique ne fait en réalité pas beaucoup de différence. Cela va uniquement modifier la syntaxe à utiliser pour gérer des scénarios un peu plus complexes.
Essentiellement, la seule différence entre les trois frameworks au niveau du routing se situe au niveau de la syntaxe utilisée. De plus, cette différence reste très légère.

Partage de code

Une des problématiques qui revient très fréquemment lors du développement d'une application est le partage de code. Avec des composants, nous créons des portions de logique associées à une vue et celles-ci peuvent être réutilisées à plusieurs endroits dans notre application. Au bout d'un moment, particulièrement lorsque notre application atteint une taille conséquente, cela n'est plus suffisant pour pouvoir organiser correctement notre code.
Chacun des frameworks offre la possibilité de partager du code à travers différents composants.
Avec Angular, cela passe par la notion de service.
Un service est représenté par une classe Typescript. Son objectif est justement d'héberger du code qui ne serait pas spécifique à un seul composant.

```
@Injectable({
  providedIn: 'root',
})
export class ProductService {
  products$: Subject<Product[]> = new BehaviorSubject<Product[]>([]);

  addProduct(product: Product) {
    this.products$.next([this.products$.value, product]);
  }
}
```

Une fois défini et configuré, Angular pourra injecter ce **service** au sein de n'importe quel composant de notre application. Pour cela, on définit un argument dans le constructeur de notre composant qui match notre service (c'est le type de l'argument que va aller regarder Angular).

```
export class AppComponent {
  products$: Subject<Product[]>;

  constructor(public productsService: ProductsService) {
    this.products$ = this.productsService.products$;
```

```
  }

  addProduct(product: Product) {
    return this.productsService.addProduct(product);
  }
}
```

On peut se demander pourquoi s'embêter avec un système d'injection de dépendances alors qu'on aurait pu par exemple créer une simple fonction, l'exporter et l'utiliser dans notre composant.
Un service est un singleton (si on le souhaite, et ce qui est très très souvent le cas). Autrement dit, la même instance de notre service sera partagée à tous les composants qui en ont besoin. On peut donc très facilement stocker un état dans un service et définir des traitements qui dépendent de cet état. De plus, ce mécanisme d'injection de dépendances apporte aussi plus de souplesse dans la mise en place de tests: on pourra facilement surcharger au sein d'un test - et uniquement dans ce test - ce qui est injecté.
Pour ce qui est de React, cette notion de services n'existe pas. À la place, on trouve une notion appelée **Hooks**. Un Hook, c'est une fonction qui va nous permettre de réutiliser de la logique à travers différents composants.

```
function useProducts() {
  const [products, setProducts] = useState([]);
  const addProduct = product => {
    setProducts([...products, product]);
  };

  return {
    products,
    addProduct
  }
}
```

Une règle des hooks est que la fonction doit commencer par le terme "use".

```
function App() {
  const { products, addProduct } = useProducts();
  ...
}
```

Une fois mon Hook défini, je n'ai qu'à l'importer (comme une fonction classique) et l'appeler dans mon composant. Quelle différence avec une fonction classique alors ? Un Hook est traqué par React et va être capable de conserver un état. Dans mon exemple, peu importe le nombre de fois où ma fonction composant App est appelée, la variable d'état "products" conservera la même valeur tant qu'elle ne sera pas explicitement modifiée à l'aide de la fonction "setProducts".
Enfin, Vue depuis sa version 3 présente un mécanisme fortement inspiré des hooks de React. L'avènement de Vue 3 a introduit **l'API Composition**. À travers elle, nous avons maintenant la possibilité de créer des **fonctions de composition** qui vont nous permettre de facilement partager une portion de code, là encore avec la possibilité d'avoir un état local.

```
function useProducts() {
  const products = ref([]);

  const addProduct = product => {
    products.value.push(product);
  }

  return {
    products,
    addProduct
  }
}
```

Cette fonction de composition peut ensuite être utilisée au sein d'un composant, dans une fonction appelée **setup**.

```
export default {
  setup() {
    const { products, addProduct } = useProducts();
  }
}
```

On constate donc qu'avec les trois frameworks la problématique du partage de code est couverte. Là encore, la manière de répondre à cette problématique diffère un peu entre chaque framework, mais l'important reste que le problème est adressé par des solutions qui ont maintenant fait leurs preuves.

Gestion de formulaire

La gestion de formulaires est un élément central de tellement d'applications qu'il me paraît essentiel de le citer.
Angular fournit un package dédié à cette gestion. Ce package nous donne accès à deux stratégies:

- le template-drive form
- le reactive form

Avec le template-driven form, nous allons principalement rajouter des éléments dans les templates de nos composants pour gérer nos formulaires. Avec le reactive form, nous allons explicitement construire un modèle de gestion dans notre code Typescript et l'associer aux formulaires dans nos templates.
Dans les deux cas, grâce à élément primitif d'Angular appelé 'directive', nous allons obtenir pour chaque champ un objet représentant son état. On y trouve notamment:

- value (la valeur inscrite dans notre champ)
- touched/untouched (le champ a-t-il déjà été blur au moins une fois ou non)
- dirty/pristine (la valeur du champ a-t-elle déjà changé au moins une fois ou non)
- valid/invalid (vis-à-vis des validateurs appliqués sur-le-champ)
- errors (les éventuelles erreurs sur notre champ)
- valueChanges (un flux nous notifiant de chaque changement de valeur)

Nous avons également la possibilité de récupérer ces informations non pas sur un champ spécifique, mais sur tout un ensemble de champs (par exemple, tous les champs du formulaire).
Avec ce package, Angular couvre la majorité des besoins en ce qui concerne la gestion de formulaires.

React ne propose pas de package officiel permettant de gérer facilement des formulaires. Il est tout à fait possible de récupérer la valeur d'un champ uniquement avec React, sans librairies tierces. Cela ressemblerait à quelque chose comme :

```
function App() {
  const [username, setUsername] = useState('');

  return (
    <form>
      <input
        type="text"
        name="username"
        value={username}
        onChange={event => setUsername(event.target.value)}
      />
    </form>
  )
}
```

On se retrouve alors avec ce qu'on appelle en React un **composant contrôlé** : le champ est complètement contrôlé par notre code.
Mais on se retrouve rapidement limité si nous avons besoin de gérer des scénarios plus complexes, ne serait-ce que pour afficher un message d'erreur à notre utilisateur par exemple. Il existe donc des librairies qui permettent de se simplifier la vie telle que **React-Final-Form** pour arriver à un niveau de détails similaire à ce qu'on trouve avec Angular.
En Vue, on trouve ce type spécial d'élément appelé **Directive** et notamment la directive **v-model** qui va nous permettre de relier une variable à un champ.

```
<input type="text" name="username" v-model="username">
```

De cette manière, on créé une liaison bidirectionnelle. Tout ce que va renseigner l'utilisateur dans ce champ se retrouver inscrit dans la variable "username" et si on modifie de manière programmatique cette variable, le changement sera reflété dans ce champ.
Cependant, comme avec React, on arrive très vite aux limites de ce mécanisme dès lors qu'on souhaite gérer des scénarios un peu plus complexes. Dans ce cas, il est recommandé de passer par des librairies tierces comme **vuelidate** par exemple.
En conclusion, Angular permet une gestion complète de formulaire directement avec un package officiel. Vue et React n'offre par défaut qu'un moyen de créer un lien entre une variable et un input, sans plus d'informations, mais cela est très bien compensé par des librairies tierces qui ont déjà fait leurs preuves.

Requêtes http

Enfin la dernière chose qu'on retrouve dans presque n'importe quelle application sont les requêtes http.
Avec Angular, son package **@angular/common** nous fournit un service **HttpClient**

```
constructor(private httpClient: HttpClient) {
  this.httpClient.get('https://products.api.com/products')
    .subscribe(...)
}
```

Une spécificité de ce service et que, comme beaucoup de choses avec Angular, la requête nous est représentée sous la forme d'un Observable.
Avec ce service, il est possible de configurer ce qu'on souhaite que ce soit au niveau des headers ou du body de la requête. On pourra également avoir accès à toutes les informations de la réponse, qu'elle soit en erreur ou pas.
Angular offre aussi par défaut un mécanisme d'intercepteurs qui nous permet d'affecter toutes les requêtes qu'on déclenche depuis notre application. C'est très pratique pour par exemple ajouter automatiquement un token d'autorisation dans les headers de chacune de nos requêtes.
À ce niveau-là, React est beaucoup plus bas niveau et nous invite à utiliser le client JS de notre choix pour effectuer nos requêtes http. Par exemple, aujourd'hui nous pouvons tout simplement utiliser l'api native fetch:

```
function fetchProducts() {
  return fetch('https://products.api.com/products')
    .then(res => res.json())
}
```

Rien de spécifique à React avec cette syntaxe. Une requête effectuée avec l'api fetch nous fournit une promesse, là aussi un objet natif JavaScript.
Et avec Vue c'est très simple, c'est exactement la même chose qu'avec React.
Angular présente un service spécifique qui se base sur des observables. React et Vue préfèrent laisser libre choix à leurs utilisateurs de la solution à employer pour déclencher une requête http.
Au final, nous avons le choix de la solution à employer pour déclencher une requête http, peu importe le framework. L'avantage étant que même sans outil spécifique à un framework, l'API native fetch est aujourd'hui supportée par tous les navigateurs récents (https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API).

Usage

Un dernier point qu'il me semble important d'aborder est l'usage auquel nous destinons le framework. On entend très régulièrement dire que pour un petit projet, mieux vaut tel framework, pour un projet de taille moyenne celui-ci, et pour un gros projet celui-là.
C'est faux.

Peu importe la taille de votre projet et ce à quoi vous le destinez, choisir Angular, React ou Vue ne sera pas un mauvais choix. La seule vraie contrainte à ce niveau-là qui fait sens est l'existant dans votre environnement.

Si vous avez déjà une application front que vous ne pouvez pas vous permettre de reprendre de zéro, mais que vous souhaitez y rajouter un framework pour vous simplifier les futurs développements, alors mieux vaut partir sur React ou Vue. Ils peuvent facilement s'inclure par petites touches, Angular non. Et encore, uniquement dans le cas où votre application existante n'est pas en Angular.js - si c'est le cas, Angular prévoit un mécanisme permettant de lancer du code Angular.js au sein d'une application Angular.

Cela s'explique par les différences dans la nature même d'Angular vis-à-vis de React et Vue. Angular se veut être un framework complet, permettant de gérer l'intégralité des aspects de votre application sans avoir besoin de rajouter de librairies tierces.

C'est le seul et unique cas de figure qui permet, à ma connaissance, de trancher entre Angular et le reste. Mis à part cela, Angular, React et Vue conviennent à n'importe quelle typologie de projet.

Conclusion

Finalement, j'espère vous avoir montré à travers ce dossier que rien ne distingue de manière définitive les trois frameworks. Que ce soit dans leurs conditions d'utilisations, leurs langages, leurs maturités, leurs communautés, leurs performances ou encore les fonctionnalités qu'ils offrent (avec ou sans solution officielle). Seules leurs syntaxes, leurs mécanismes internes, leurs façons de proposer des solutions diffèrent de l'un à l'autre.

Malgré cela, ils partagent énormément de concepts. Ce qui fait que les trois frameworks sont tous des choix pertinents pour le développement d'une application web.

Cela explique pourquoi à la question "quel framework JS choisir ?", la réponse est si souvent "ça dépend". Pour sortir de cette réponse, voici quelques questions que je me pose au moment de devoir choisir, que je trierai en 3 catégories.
La question qui va obligatoirement orienter vers un framework plutôt qu'un autre :

- Est-ce que je crée une application from scratch ou dois je m'inclure dans une application JS existante ou non ? Si oui, est-ce une application Angular.js ?

Préférez utiliser React ou Vue si vous ne souhaitez gérer à travers un framework qu'une petite partie d'une application JS, et que cette dernière n'utilise pas Angular.js.
Les questions essentielles qu'il vaut mieux se poser :
Quelles sont les expériences et les compétences déjà présentes dans mon équipe ?

Est-ce que l'un de ces frameworks a déjà été utilisé quelque part dans mon entreprise ?

Si votre équipe a déjà de l'expérience et des compétences sur un de ces frameworks, n'hésitez pas à partir sur celui-ci. Si une application déjà existante dans votre entreprise se sert d'un de ces frameworks, alors c'est là aussi un critère tout à fait pertinent pour choisir un framework et rester cohérent avec la direction technique globale.
Et finalement les questions qu'on va tous se poser, mais qui n'ont finalement que peu d'importance :

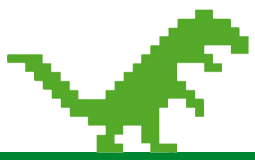
- Mon application est censée faire ça, je dois prendre un framework en particulier ?
- De quelle manière je préfère coder ?

L'usage de votre application n'a que très peu d'importance, voire pas du tout. Dans la majorité des cas, peu importe ce que votre application devra faire, les 3 frameworks vous mettront à disposition tous les outils nécessaires pour le réaliser.

Suis-je plutôt adepte de la programmation fonctionnelle ou de celle orientée objet ? Est-ce que j'aime la flexibilité du JavaScript où la sécurité apportée par le Typescript ? Cette question sur la manière de coder, vous vous la poserez forcément si vous devez choisir un framework. Ce n'est pas une mauvaise chose. Je souhaite simplement insister sur le fait que ce n'est pas la question qui doit orienter votre choix en priorité.

Si après avoir répondu aux questions précédentes, aucun framework ne sort du lot, alors il est tout à fait OK de faire son choix en fonction de ses préférences de programmation. Avec uniquement mes quelques années d'expérience dans le monde du développement JS, j'ai déjà vu un bon nombre de projets dont le framework a été choisi sur ce critère - et qui fonctionnent !

Ce qu'il faut retenir, c'est qu'à partir du moment où les concepts sont assimilés, peu importe que vous choisissiez Angular, React ou Vue, vous vous en sortirez très bien :)



TECHNOSAURES

www.technosaures.fr

Le magazine à remonter le temps !

Comment sécuriser son application JavaScript en 2022 ?

JavaScript, connaît un succès populaire, notamment dans le développement web. À l'origine, c'est un langage de scripting léger. Certains de ses détracteurs m'ont interpellée avec véhémence : JavaScript pose de sérieux problèmes de sécurité. Certaines fonctionnalités natives sont dangereuses, l'exécution côté serveur peut compromettre tout le système. Le propos de cet article est d'exposer des moyens de sécurisation utilisables pendant tout le cycle de vie d'une application full-stack JavaScript.

Romy Alula

Consultante en développement Web chez Codeworks. Membre active de Ladies of Code et Duchess France, elle y accompagne de jeunes devs en tant que marraine. Sur des sujets comme le Craft, la préparation d'entretien technique ou encore la négociation. Depuis mars 2021, elle anime le hands-on sur le TDD, chez les Ladies of code. Guidée par sa passion et sa curiosité, elle partage ses apprentissages et découvertes avec d'autres curieuses et curieux au travers de coding dojos, meetups et autres conférences. Dès qu'elle a du temps libre, elle joue du piano et encourage ses enfants à exprimer leur créativité. @Goumies

En 2019, le State of the Software Supply Chain de Sonatype constatait que plus de 80% du code des solutions modernes provient des dépendances. D'après l'édition 2020, près de 40% des paquets NPM reposent sur du code contenant des vulnérabilités connues.

Les éditions 2017 et 2021 de l'OWASP top 10 remontent le risque de baser son projet sur des dépendances vulnérables. Le couplage commence avec les dépendances. Comme dit Michael Feathers : "la gestion des dépendances est un des problèmes les plus critiques dans le développement logiciel".

Réduire la surface d'attaque en limitant le nombre de dépendances

Le premier enjeu est donc de bien définir les dépendances. Avant chaque nouvel ajout au projet, il est bon de se renseigner sur le module et ses alternatives. Sur NPM trends (www.npmtrends.com), les principales statistiques des packages permettent de faire un choix plus éclairé.

Dans un souci de stabilité, limiter le nombre de dépendances amoindrit la vulnérabilité de votre solution. Une dépendance embarquant ses dépendances transitives. Chaque dépendance doit être régulièrement mise à jour. Pour en assurer le bon déroulement, figer les versions dans le `package.json` en supprimant les "matchers" comme `^`, `~`, `<` ou `>` et en versionnant le `package-lock.json`. Cela devrait réduire le risque de voir planter le build du projet, au cas où la mise à jour d'une dépendance (ou d'une de ses dépendances transitives) la rendrait incompatible.

Lockfile-lint - passe en revue les fichiers `lock` pour une meilleure sécurité et conforter les politiques de sécurité pour parer l'injection de packages malicieux et autres configurations faillibles (https://github.com/lirantal/lockfile-lint).

`npm audit` ou snyk permet, par exemple, de détecter les librairies qui ne sont plus à jour.

La commande `npm-shrinkwrap` génère un fichier, npm-shrinkwrap.json. Cette pratique est recommandée uniquement pour les applications déployées via le processus de publication dans le registre NPM : comme les daemons, les outils de lignes de commandes installés globalement ou encore les devDependencies.

Attention au typosquatting

Risque propre à tout gestionnaire de paquets avec des modules non vérifiés. Le typosquatting consiste à tirer profit de la notoriété d'un package pour diffuser des librairies malicieuses. L'incident "crossenv" embarquait les mêmes fonctionnalités que "cross-env" et envoyait les variables d'environnement vers un serveur distant.

(sources: https://snyk.io/blog/typosquatting-attacks, https://adtmag.com/articles/2020/10/01/sonatype-catches-typosquatters.aspx, https://thenewstack.io/npm-cleans-typosquatting-malware)

Le typosquatting existe dans d'autres registres. Que ce soit PyPi ou Rubygems. L'open source offre des solutions pour toutes les facettes du développement.

La sécurité open source

NPM Public Advisories centralise les données liées aux failles de sécurité dans l'écosystème JavaScript. Le Node advisories navigator permet de parcourir les vulnérabilités antérieures à 2019 en fonction du type de vulnérabilité. Et ce, à des fins éducatives.

CONCEPTS-CLÉS

SAST (Static Application Security Testing) : analyse statique du code source pour y détecter des patterns de vulnérabilité. Cela peut provoquer de nombreux faux positifs.

DAST (Dynamic Application Security Testing) : exécution de tests d'intrusion automatisés, dans une sandbox. Au runtime, scanne des réponses à une variété de payloads pour trouver des défauts de sécurité.

IAST (Interactive Application Security Testing) : technique d'analyse mélangeant les SAST et DAST. Elle a l'avantage d'analyser des routes de codes réelles (exécutées en production ou pendant les tests). Ce qui limite les faux positifs. Elle n'est pas autant répandue que les deux autres.

AST (Abstract Syntax Tree) : Arbre de la Syntaxe Abstraite construit après l'analyse du code. C'est la base utilisée par le linter pour trouver les erreurs. https://fr.wikipedia.org/wiki/Arbre_de_la_syntaxe_abstraite

CSP (Content Security Policy) : https://developer.mozilla.org/fr/docs/Web/HTTP/Headers/Content-Security-Policy

Forensic Analysis : Science forensique : La criminalistique est l'ensemble des techniques mises en œuvre par la justice, la police et la gendarmerie pour établir la preuve d'un délit ou d'un crime et en identifier son auteur. https://enterprise.comodo.com/blog/what-is-forensic-analysis/, https://www.virtualmetric.com/blog/forensic-analysis-and-the-security-of-infrastructure

Surface d'attaque : Somme de tous les vecteurs d'attaque qu'une personne peut utiliser pour s'introduire dans une application ou un système. https://fr.wikipedia.org/wiki/Surface_d'attaque

Attaquant.e : individu ou groupe d'individus, généralement mal intentionné(s) qui recherche(nt) et tente(nt) d'exploiter les failles de sécurité.

Âge de la vulnérabilité : durée de vie depuis l'introduction de la vulnérabilité dans le code. Plus la vulnérabilité est "âgée", plus grand est le risque d'exploitation.

Déni de service : Dos (Denial of Service), attaque consistant à faire planter une application, que ce soit via la pollution de prototype ou encore une regex malicieuse. Une intervention manuelle est indispensable pour y remédier. L'application est donc indisponible pendant plus ou moins longtemps.

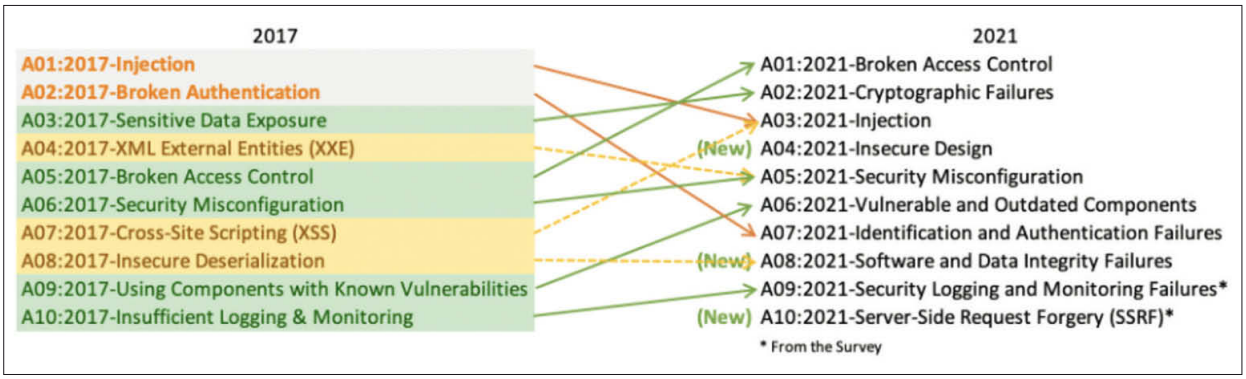


Figure 1

OWASP

OWASP (Open web Application Security Project) est une communauté en ligne, à but non lucratif. Son travail se focalise sur l'amélioration de la sécurité du logiciel.

Le meetup OWASP France Chapter avec Yvan Phélizot & Didier Bernaudeau (https://www.meetup.com/fr-FR/owasp-france) propose un événement mensuel. Les participant.e.s peuvent présenter ou découvrir des sujets, en fonction du thème de l'édition.

L'un des projets phares de l'OWASP, l'OWASP Top 10, documente les risques de sécurité les plus critiques. Sa version française est disponible : https://owasp.org/Top10/fr. C'est un bon point de départ pour se familiariser aux problématiques de sécurité.

FAILLES DE SÉCURITÉ

OWASP 2021

L'édition 2021 du top 10 a été publiée quelques semaines avant que je finalise la rédaction de cet article. **Figure 1**

Sur la base de la présentation de Lewis Arden, consultant en sécurité, passons en revue l'OWASP Top 10 pour l'écosystème JavaScript. Le projet "insecure-code-test" nous servira de support. Il s'agit d'une version peu sécurisée de l'application web permettant d'évaluer les candidats pendant les workshops techniques.

A01:2021-Ruptures de contrôles d'accès

Face à des restrictions non renforcées, les attaquants accèdent à des données et fonctionnalités sans autorisation : accès aux comptes d'autres utilisateur.trice.s, à des fichiers sensibles, modification de données, changement de droits...

Pour renforcer l'authentification, l'autorisation, et la logique Métier, il existe des solutions côté serveur :

- Node Casbin, gestionnaire de contrôle d'accès basé sur les rôles (Role-Based Access Control - RBAC) et attributs (Attribute-based access control - ABAC)
- Sails.js, https://sailsjs.com, framework
- Créer des middlewares personnalisés avec des JWT (JSON web Token)
- OpenID-client, https://www.npmjs.com/package/openid-client, basé sur OpenID Connect et OAuth2

A02:2021-Défaillances cryptographiques

Il existe des librairies de chiffrement. Spécialisées et maintenues par des experts. Elles sont à privilégier pour stocker des données sensibles comme les mots de passe. La défaillance cryptographique peut compromettre tous les comptes d'une application. L'attaquant accède, simplement, à une base de données où les mots de passe sont stockés avec un salt unique. Ou pire, ils sont stockés en clair.

Il est indispensable de bien choisir les algos de chiffrement ou de hash et de les utiliser correctement. Le standard de l'industrie est `bcrypt`. Il embarque la génération du salt. Et permet de confirmer la correspondance d'un mot de passe sans exposer le secret. Écrire son propre algo de chiffrement ou réimplémenter un algo connu est fortement déconseillé.

Dans le fichier `routes.js`, la création d'utilisateurs s'appuie sur les contrôleur et service responsables.

```
<pre>
fastify.post("/user", async (request, reply) => {
 validate(request.body, reply);

 const foundCodeworker = await findAlreadyRegistered(
  request.body.codeWorkerEmail,
  codeworkers
 );
 checkEmail(foundCodeworker, reply);

 const createdCodeWorker = await save(request.body, codeworkers);
 succeedOnCreation(createdCodeWorker, reply);
});
</pre>
```

Dans le fichier `createUser.js`, chaque nouvel utilisateur est persisté en base de données MongoDB.

```
<pre>
const { hash } = require("../../../infrastructure/webserver/utils/bcrypt");

module.exports = {
 findAlreadyRegistered: async (email, codeworkers) => {
  return await codeworkers.findOne({
   email,
  });
 },
 save: async (user, codeworkers) => {
  return await codeworkers.insertOne({
   email: user.codeWorkerEmail,
   password: await hash(user.codeWorkerPassword),
   authority: "user",
  });
 },
```

```
};
</pre>
```

Dans le fichier `logUserIn.js`, `checkPassword` confirme la correspondance entre les mots de passe.

```
<pre>
const { verify } = require("../../../infrastructure/webserver/utils/bcrypt");

module.exports = {
 find: async (email, codeworkers) => {
  return await codeworkers.findOne({
   email,
  });
 },
 checkPassword: async (userPassword, foundCodeworkerPassword) => {
  if (!foundCodeworkerPassword) return false;
  return await verify(userPassword, foundCodeworkerPassword);
 },
};
</pre>
```

`bcrypt` fournit les fonctions :
- `hash` pour calculer une chaîne de caractères aléatoires que l'on stocke
- `compare` pour comparer le hash stocké et le mot de passe en clair saisi dans le formulaire

```
<pre>
const bcrypt = require("bcrypt");

module.exports = {
 hash: (password) => {
  return new Promise(async (resolve, reject) => {
   await bcrypt.genSalt(10, async function (err, salt) {
    if (err) reject(err);
    await bcrypt.hash(password, salt, function (err, hash) {
     if (err) reject(err);
     resolve(hash);
    });
   });
  });
 },
```

```
 verify: (password, hash) => {
  return new Promise(async (resolve, reject) => {
   await bcrypt.compare(password, hash, function (err, result) {
    if (err) reject(err);
    resolve(result);
   });
  });
 },
};
</pre>
```

A03:2021-Injection

Les dangers du mélange de données et de code. On peut penser aux requêtes SQL, ou encore aux logs. Dans l'écosystème JavaScript, on pourra s'intéresser à l'injection en NoSQL.

L'absence d'injection SQL ne veut pas dire qu'il n'y a pas d'injection en NoSQL.

L'attaque consiste à envoyer des données hostiles pour piéger l'interpréteur. Le recours aux sélecteurs de requête de MongoDB peut provoquer de sérieux dégâts. Rendus possibles par l'exécution indésirable de commandes ou l'accès aux données sans autorisation valide.

Le code prenant directement en entrée la saisie venant du client est vulnérable si :

La saisie inclut un sélecteur de requête Mongo: $ne, $lt, $gt, $eq, $regex…

La saisie est directement passée en paramètre d'une méthode de collection: find, findOne, findOneAndUpdate…

L'inclusion directe dans une méthode de collection, comme `find` ou `findOne`. **Figure 2**

Ici, l'injection NoSQL de l'opérateur `$ne` (not equal) ne permet pas de s'authentifier. Par contre, elle permet de retourner le premier utilisateur stocké dans la base de données. **Figure 3**

Pour parer à ces attaques, on peut utiliser un schéma ou créer un validateur personnalisé avec notamment Joi (https://www.npmjs.com/package/joi).

Code complet sur programmez.com & github

Source : documentation de Joi, https://joi.dev/api

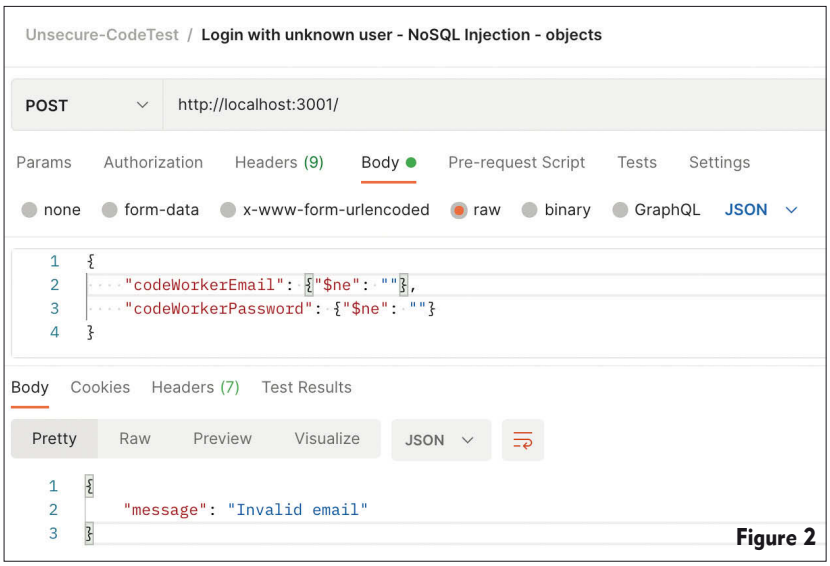


Figure 2

```
foundCodeworker : {
  "_id": "61d9a7651692411d4e8af342",
  "email": "coddyOne@codeworks.fr",
  "password": "Dummy",
  "communities": [
    "JS",
    "JVM"
  ],
  "roles": [
    "mentor",
    "writer",
    "speaker"
  ],
  "authority": "admin"
}
```

Figure 3

A7:2017 Cross-Site Scripting (XSS)

Le XSS est facile à introduire.

```
<pre>
const userName = location.hash.match(/userName=([^^&]*)/)[1]
// ...
div.innerHTML += `Welcome ${userName}">`
</pre>
```

Exécution du script:
`http://www.vulnerable.site/#userName=<img src=malicious.site onerror='alert(document.domain)'>`

La façon la plus simple de s'en prémunir reste encore de se méfier de toutes les entrées utilisateurs : échapper ou faire de la sanitization. Le XSS DOM est dur à contrer dans l'écosystème actuel. Chaque navigateur analyse et rend le HTML différemment. Divers contextes d'exécution et encodages coexistent. Utiliser les APIs sécurisés est un bon moyen d'éviter ce type de faille :

- préférer `innerText`à `innerHTML`
- encodeURI

Parmi les frameworks dédiés à la désinfection d'entrée par design, on peut citer :

- xss
- sanitize-html
- dompurify

Comprendre les risques encourus est primordial pour s'en servir correctement.

Une autre façon de contrer le XSS, c'est de recourir à une CSP (Content Security Policy) ou stratégie de sécurité du contenu. Il s'agit du header HTTP `Content-Security-Policy` ajouté aux réponses du serveur.

A05:2021-Mauvaise configuration de sécurité

Problème très répandu, la mauvaise configuration liée à la sécurité. C'est souvent le résultat d'une configuration par défaut non sécurisée, incomplète ou ad hoc. On peut ajouter à cette liste : le stockage cloud ouvert, la mauvaise configuration des headers HTTP ou encore des messages d'erreur verbeux contenant des informations sensibles. L'OWASP propose un projet pour se familiariser avec la sécurisation des headers : Secure headers, https://owasp.org/www-project-secure-headers.

Un environnement Node bien paramétré en production avec la variable d'environnement `NODE_ENV=production` permet, notamment, d'éviter ce dernier point. À cela peuvent s'ajouter les vérifications dans le middleware. Il s'agit alors de s'assurer que les scripts ne sont pas exécutés avec des droits d'administrateur.

A06:2021-Composants vulnérables et obsolètes

Quand une faille connue est identifiée dans un repo Github, Dependabot (bot natif de Github) y ouvre des Pull Request. Les contributeurs n'ont plus qu'à accepter la mise à jour.

La vigilance reste de mise. Le 4 novembre 2021, la librairie "coa" a servi de point d'entrée pour une tentative de vol massif de mots de passe stockés dans les navigateurs Chrome (sources: https://www.bleepingcomputer.com/news/security/popular-coa-npm-library-hijacked-to-steal-user-passwords, https://github.com/veged/coa/issues/99). Ce dépôt, dont la dernière mise à jour officielle remonte à 2018, a été actualisé avec plusieurs publications dans la même journée. Bien que suspecte, cette activité a eu des répercussions sur des millions de projets.

Les incidents se produisent aussi sur les packages les plus populaires: express, lodash (développé plus loin avec la pollution de prototype), hapi, jquery. NPM permet de repérer et réparer les vulnérabilités connues au sein des packages installés. La commande `npm audit` liste les défaillances et leur niveau. Lancer `npm audit fix` permet de mettre à jour automatiquement la librairie, quand c'est possible.

L'OWASP propose des outils de détection des dépendances avec des vulnérabilités connues :

- OWASP Dependency Check - identifie les dépendances et vérifie l'existence de vulnérabilités publiées
- Retire.JS est un scanner de librairies JavaScript

Sonatype propose un outil de test d'application automatisé, le Nexus Vulnerability Scanner (NVS).

A07:2021-Identification et authentification de mauvaise qualité

La comparaison non sécurisée d'objets peut engendrer une authentification trompeuse.

Tous les objets héritent de la classe Object en JavaScript. Vérifier les propriétés natives comme `constructor` ou les méthodes comme `hasOwnProperty` n'est pas suffisant. Il est recommandé de faire appel à une API ou librairie spécialisée, en matière d'authentification.

Node dispose d'une API, Crypto: `crypto.timingSafeEqual(a, b)` neutralise les attaques basées sur le temps (ou "timing attacks").

Identifier et authentifier restent exigeant en termes de bonnes pratiques de développement et développement sécurisé. C'est pourquoi utiliser OAuth2 au bénéfice d'une authentification basique est fortement recommandé.

A09:2021- Carence des systèmes de contrôle et de journalisation (A10:2017 - Insufficient Logging & Monitoring)

Là encore, l'OWASP diffuse du contenu instructif. Ces deux "antisèche" nous guident pour contrer le risque d'insuffisance de logging et monitoring :
https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html
https://cheatsheetseries.owasp.org/cheatsheets/Nodejs_Security_Cheat_Sheet.html#handle-uncaughtexception

LES TESTS DE SÉCURITÉ APPLICATIVE

DAST : OWASP Zap

J'ai découvert ZAP (Zed Attack Proxy) dans le cadre du workshop "Become a hacker" de Paul Molin, web Application Security Evangelist chez Theodo.

ZAP est proxy local installé sur votre machine. Ce qui permet à ZAP de générer un arbre représentant l'application analysée.

La liste des fonctionnalités regroupe :

- Proxy, toutes les requêtes et réponses passent par ZAP
- Passive scanning, toutes les requêtes et réponses sont analysées pour détecter les vulnérabilités
- Spidering, toutes les réponses sont "parsées" pour découvrir tout nouveau contenu
- Active scanning, toutes les requêtes sont rejouées pour identifier les plus grandes vulnérabilités
- Fuzzing, les requêtes sont aléatoirement modifiées
- Scripts, pour étendre les fonctionnalités

Vous pouvez retrouver les points-clé du workshop sur ce blog : https://blog.theodo.com/2016/04/how-to-become-a-hacker-in-10-minutes.
Il existe aussi une CLI pour brancher ZAP à votre usine logicielle (https://www.zaproxy.org/docs/desktop/cmdline).

SAST : Node Secure

Nsecure - Node.js CLI that allow you to deeply analyze the dependency tree of a given npm package or a directory, github node-secure
Dans les pages suivantes, nous reviendrons sur Nsecure et la sécurité de l'écosystème de Node.js dans le cadre de l'interview de Thomas Gentilhomme. Mentor de devs, il est aussi membre du Node.js security working group.
Nous avons tout un panel d'outils à notre disposition pour nous alerter en cas de risques détectés. La façon la plus simple de les éviter c'est de bien connaître le langage pour nous passer de ses fonctionnalités qui sont des vecteurs d'attaque.

Les fonctionnalités dangereuses de JavaScript

Bannir les comparaisons faibles avec le double égal pour leur préférer la stricte égalité ou `Objetc.is(firstObject, secondObject)`
Éviter les fonctions dangereuses comme `eval(alert('Alert !'))` qui acceptent le code sous forme de chaîne de caractères et l'exécutent.
https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html#rule-3-javascript-encode-before-inserting-untrusted-data-into-javascript-data-values
Invoquer le mode strict avec "use strict" (sauf en cas de rétrocompatibilité antérieure à l'ES5 requise) lève des erreurs en cas de "mauvaise syntaxe".
Pour éliminer toutes les fonctionnalités dangereuses, ESLint repère les anti-patterns dans le code. Ajouter ces quelques règles au fichier `.eslintrc.json` :

- "no-eval": intercepte tous les appels directs à la fonction `eval`
- "no-implied-eval": avertit en cas d'utilisation des variantes risquées de `setTimeout` et `setInterval`
- "no-new-func": empêche le code d'utiliser la fonction `constructor`
- "eqeqeq": force l'utilisation de l'égalité stricte dans toutes les situations, sauf si vous optez pour une gestion spécifique de l'égalité avec `null`

```
<pre>
{
 "env": {
  "browser": true,
  "commonjs": true,
  "es2020": true
 },
 "parserOptions": {
  "ecmaVersion": 11
 },
 "extends": "eslint:recommended",
 "rules": {
  "no-eval": "error",
  "no-implieno-implied-eval": "error",
  "no-new-func": "error",
  "eqeqeq": [eqeq": ["error", "always"]
 }
}
</pre>
```

Pour vérifier l'égalité stricte entre objets, préférez `Object.is`. Cette fonction fait une comparaison stricte, sauf dans des cas à la marge, en ce qui concerne les nombres.
Il est recommandé de toujours vérifier le type des données provenant du client. Des packages de validation et désinfection de formulaires bloqueront toutes tentatives d'altération du comportement de l'application.
Pour renforcer la sécurité de votre code source, il existe des packages dédiés référencés dans la liste awesome Node.js security (Nodejs Security Cheat Sheet, do not use dangerous functions) :

- le plugin de sécurité d'eslint : https://www.npmjs.com/package/eslint-plugin-security
- js-x-ray, le scanner SAST pour JavaScript et Node.js, capables de détecter divers patterns de code malicieux bien connus (*) : https://github.com/fraxken/js-x-ray

(*) import non sécurisé, instruction non sécurisée, regex non sécurisée, littéraux encodés, codes minifiés et offusqués
Dans la continuité des points d'attention propres à notre langage, la pollution de prototype est à garder en tête dans les interactions entre le client et le serveur.

Prototype pollution : de quoi parle-t-on ?

La chaîne de prototypes : particularité de JavaScript
L'une des caractéristiques principales des langages orientés objet est l'héritage. L'héritage est construit par les relations entre les classes(*). JavaScript repose sur la chaîne des prototypes. Tout objet peut être à la fois objet et prototype. Chaque objet possède la propriété `__proto__` qui retourne son prototype (un objet entier).
Les objets ont leurs propres propriétés et des propriétés héritées grâce à la chaîne de prototypes. Les propriétés propres à l'objet sont déclarées à la création de l'objet. Elles peuvent aussi être ajoutées lors de l'exécution du code. Les propriétés héritées qui, quand elles sont définies, ne sont modifiées que pour l'objet propriétaire. La chaîne de prototypes est donc dynamique.
Sa nature dynamique permet de modifier un objet de la chaîne, n'importe quand. Ce qui peut rendre le code d'une application vulnérable aux attaques de pollution de prototype (prototype pollution).
Par le passé, l'altération des méthodes natives de JavaScript a pu entraîner un déni de service : envoyer une chaîne de

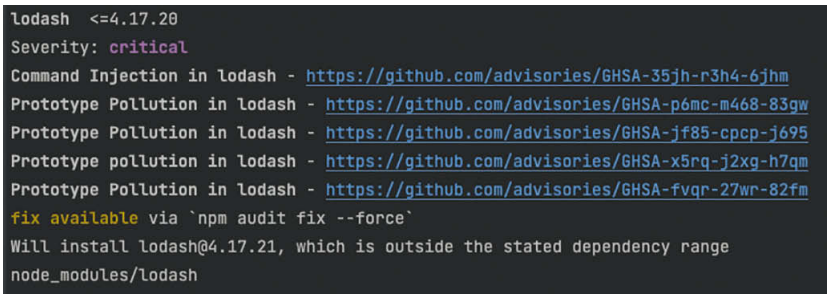


Figure 4

caractère là où une fonction était attendue suffisait à lever une erreur, avec la version 4.17.4 de lodash. La navigation d'une application web était alors rendue impossible : toutes les pages affichaient `TypeError: Object.prototype.toString .call is not a function`. Pour remédier à cette vulnérabilité, il fallait installer une version de lodash plus mature en termes de sécurité. **Figure 4**

—

(*) Les classes apparues en JavaScript avec l'ES6 ne sont que du sucre syntaxique pour définir la chaîne de prototypes. JavaScript utilise toujours l'héritage prototypal.
(source: https://portswigger.net/daily-swig/prototype-pollution-the-dangerous-and-underrated-vulnerability-impacting-javascript-applications)

—

La fixation de session (Session fixation) consiste à ajouter les identifiants d'un compte légitime à la chaîne de prototype. En l'absence d'identifiants dans la requête de connexion, les identifiants usurpés peuvent être envoyés au serveur. L'attaquant a alors accès aux données confidentielles du compte piraté.

Pour éviter d'introduire une vulnérabilité à cette pollution, quelques bonnes pratiques :

- Geler les propriétés avec `Object.freeze (Object.prototype)` :
- Procéder à la validation des saisies JSON en adéquation avec le schéma défini
- Éviter d'utiliser des fonctions de fusion récursives de manière non sécurisée
- Utiliser des objets sans prototype tel que `Object.create(null)` pour éviter d'affecter la chaîne de prototypes
- Préférer `Map` à Object`
- Mettre régulièrement à jour les nouveaux patchs des librairies installées

Pour contrer toutes attaques de pollution de prototypes, il existe des packages dédiés référencés dans la liste awesome Node.js security :

- bourne : https://github.com/hapijs/bourne
- secure json parse : https://github.com/fastify/secure-json-parse

Monkey patching

Dans le cadre d'un BBL sur les tests d'approbation et les tests de caractérisation, je présentais mon exemple de test d'approbation manuel. Un des participants a alors évoqué le "monkey patching", ce qui a piqué ma curiosité. Mon test manuel avait pour unique objectif de rediriger la sortie de l'application. Le monkey patching, pratique déconseillée, dans la majorité des cas, ne répondait pas au besoin (source: https://www.audero.it/blog/2016/12/05/monkey-patching-javascript/).

Secure TDD

Et s'il existait une méthode de développement qui permettrait de prévenir les vulnérabilités ? Un TDD de la sécurité ?

Ce n'est pas du TDD à proprement parler. C'est surtout une méthode de tests automatisés branchés à l'usine logicielle. Son usage est recommandé en staging ou QA, déconseillé pour la prod. Le principe est de réaliser des tests automatisés, à l'aide d'outils d'analyse en tout genre :

- Statique :SonarQube, downloads - compatible avec 15 langages
- Dynamique : OWASP Zap - pratique pour les débutants en sécurité principalement focalisé sur le Top 10 OWASP, utilise Selenium en interne pour la navigation (spidering de site), paramétrable avec les APIs de ZAP, WireShark - standard de l'industrie, libre et gratuit. Les deux solutions proposent les options in-scope et out-of-scope pour limiter le périmètre du scan à certaines URL
- Forensique : Splunk - data leak, particulièrement efficace pour l'analyse des logs, permet de créer des dashboards

Secure by design

Le concept du "Secure by design", c'est de mettre la qualité au service de la sécurité. Les pratiques du Craft (ou Software Craftsmanship) comme le pair-programming, le TDD et le Clean Code viennent sécuriser l'application. Les développeurs questionnent le Métier sur les cas extrêmes, les limites des cas d'utilisation. Le but étant de renforcer le cœur de métier et les fonctionnalités qui génèrent des revenus.
(sources: https://cotonne.github.io/agile/security/craft/secure-coding/secure-by-design/2018/06/07/crafting-secure-software.html, https://fr.slideshare.net/YvanPHELIZOT/crafting-secure-software-dddeu-2019)

Pour aller plus loin

Livres en anglais : "Building secure and reliable systems", "Thinking Security: Stopping Next Year's Hackers"
Articles en anglais : "7 places to do automated Security Tests", "Security is everybody's job [1/6]"

Conclusion

Si vous souhaitez vous familiariser avec la cyber sécurité, je vous recommande Cyber security for beginners de Raef Meeuwisse. Nina Cercy, experte en cybersécurité, nous l'a conseillé, lors d'un webinaire organisé par l'Ada Tech School ("Cybersécurité : les métiers du futur ?", disponible sur YouTube). Elle affirmait que ce livre permettait d'obtenir une structure mentale puissante sur le sujet.

La sécurité dans l'écosystème JavaScript est un vaste sujet. Sujet qui pourrait remplir, à lui seul, un numéro entier de Programmez!. J'espère que cette vue d'ensemble d'une sélection de moyens de sécuriser votre appli JS vous aura donné envie d'en apprendre toujours plus. Et surtout, de pouvoir les mettre en pratique.

MON SECRET POUR DU JAVASCRIPT PLUS FIABLE ?

L'assaisonner à la sauce fonctionnelle

Laurent Bossavit

Et si je vous vendais une solution "magique" pour rendre votre code JavaScript plus fiable ? Vous demandez à voir, et vous avez raison. Dans notre métier, il faut être sceptique. Et si je vous dis que la solution consiste à… écrire dans un autre langage ? Là vous commencez à sérieusement voir un loup… ou plus probablement vous avez déjà raccroché et ajouté mon numéro à la liste noire.

Et pourtant ! Si vous êtes encore là, cet article va vous montrer que quelques-uns des concepts-clés de la programmation fonctionnelle qui se déploient avec bonheur dans des langages comme Haskell ou Lisp, se transposent de façon tout à fait efficace en JavaScript qui a, entre autres vertus, celle d'être un langage "multiparadigme". En effet, contrairement à Java qui a toujours l'étiquette d'un langage "pur objet" (quand bien même ce n'est plus tout à fait vrai à l'heure de JDK 17), JavaScript est capable de retourner sa veste de tous les côtés, objet si tel est votre désir, mais capable de vous montrer aussi sa facette "pur fonctionnel" pour peu qu'on le lui demande gentiment.

Ayant lâché le nom de Haskell, je tiens à vous rassurer : cet article est garanti 100% sans tutoriel sur les monades et restera très accessible en examinant un petit exercice de code emprunté à l'excellente plateforme Codingame et en le traitant à la sauce fonctionnelle à base de quelques ingrédients passe-partout : map, reduce, filter et (seule concession à l'exotisme) les monoïdes, concept beaucoup plus simple à expliquer qu'il ne l'est à épeler.

Le problème

Notre plat de résistance est un problème très simple, peut-être tellement simple que vous craindrez même de rester sur votre faim ! Mais pas d'inquiétude, même des choses simples en apparence peuvent en réalité cacher des subtilités.

Il s'agit d'un des premiers exercices du parcours proposé par Codingame. Ces exercices suivent un schéma invariable : votre programme, que vous pouvez écrire dans un langage de votre choix parmi une large palette, doit interpréter des entrées qui lui sont présentées sur le flux stdin (entrée standard), résoudre le problème proposé pour ces entrées puis émettre la solution au problème sur le flux stdout (sortie standard). Des jeux de test, toujours bien conçus et assez étoffés, sont là pour vous assurer que vous n'avez pas commis d'impair.

Le problème spécifique que nous examinons est intitulé « Températures » et consiste à analyser une série d'entiers (positifs ou négatifs) puis à restituer celui d'entre eux qui est « le plus proche de zéro ». Cette notion comporte une petite ambiguïté : entre 5 au-dessus de zéro et 5 en dessous de zéro, quelle est la température la plus proche ? Mais comme c'est un exercice volontairement très facile, l'ambiguïté est levée dans l'énoncé : on prendra celui des deux qui est positif.

Les exercices de Codingame se veulent accessibles aux personnes les plus novices en programmation qui, par exemple, peuvent ne pas maîtriser les API pour lire sur l'entrée standard ou convertir des chaînes de caractères en nombres. La plateforme fournit donc pour chaque exercice un canevas de code pré-écrit.

```
const n = parseInt(readline()); // the number of temperatures to analyse
var inputs = readline().split(' ');
for (let i = 0; i < n; i++) {
    const t = parseInt(inputs[i]);// a temperature expressed as an integer
 ranging from -273 to 5526
}
console.log('result');
```

Si on représentait la solution de cet exercice sous la forme d'une "to-do list", elle ressemblerait donc à ceci :

- découper une chaîne de caractères pour obtenir une liste
- transformer cette liste de chaînes en liste d'entiers
- définir une notion de "distance à zéro"
- départager deux nombres de même valeur et de signe opposé
- extraire de notre liste d'entiers celui qui minimise cette distance à zéro
- afficher le résultat

Quasiment la moitié du problème est déjà résolue pour nous, ça va donc être du gâteau.

Une première solution et ses défauts

On va donc assez naturellement se couler dans ce moule confortable, accueillant et familier comme une paire de charentaises pour émettre une première solution qui ressemblera à ceci :

```
const n = parseInt(readline()); // the number of temperatures to analyse
var inputs = readline().split(' ');
var closest = Number.MAX_VALUE;
if (n == 0) {
    closest = 0;
}
for (let i = 0; i < n; i++) {
    const t = parseInt(inputs[i]);// a temperature expressed as an integer
 ranging from -273 to 5526
    if (Math.abs(t) <= Math.abs(closest)) {
        if (Math.abs(t) != Math.abs(closest) || t > closest) {
```

```
        closest = t;
      }
    }
}
console.log(closest);
```

Ce code vous paraîtra peut-être de bon aloi : simple, sans détour, facile à lire. La simplicité de l'exercice transparaît dans le peu de lignes de codes qu'il est nécessaire d'ajouter pour résoudre le problème.

Transposez cependant cette façon de coder à une véritable appli en production et vous irez au-devant de sérieux problèmes de dette technique ! En réalité, d'une part, cette façon de coder n'est pas maintenable et ne peut pas vous amener à quelque chose de soutenable à long terme, d'autre part elle fait injure à JavaScript en n'exploitant pas les belles possibilités de ce langage.

Que reproche-t-on à ce code, vous demandez-vous peut-être ? Le principal défaut, c'est qu'il est écrit d'une façon qui a très peu de correspondance avec le problème à résoudre. Le programme ne ressemble pas du tout à notre to-do list. La conséquence, c'est qu'il faut donc un effort considérable de traduction mentale pour « remettre les pièces en place », et cet effort va être dépensé à chaque fois que nous viendrons lire ce code, pour le comprendre, pour s'en inspirer ou pour le modifier.

Une des façons, dont cela, se traduit, c'est le niveau d'imbrication des préoccupations. Dans ce code, nous avons un if imbriqué dans un autre if à son tour imbriqué dans un for. La transformation des chaînes en entier est imbriquée au beau milieu de tout ça. La notion de départage entre deux nombres qui sont à la même distance est difficilement lisible. L'initialisation à MAX_VALUE et le traitement du cas particulier de la liste vide sont côte à côte, mais c'est purement accidentel, les deux sujets n'ont rien à voir.

Un idéal platonique de solution

Voici une solution en Haskell qui me semble tutoyer la perfection. Je m'empresse d'ajouter que c'est toujours relatif et que j'ai sans doute encore des millions de choses à apprendre qui me rendent aveugle à tout un tas de défauts… et c'est la vie !

```
import Data.List (minimumBy)
import Data.Ord (comparing)

main = do
  n <- read <$> getLine
  ints <- words <$> getLine
  let temps = map read ints

  let closeToZero = (comparing abs) <> (comparing negate)
  let closest = if n == 0 then 0 else minimumBy closeToZero temps

  print closest
```

Comme promis, pas de monades, juste du code Haskell. Nous allons nous en inspirer, mais nous n'aurons pas besoin de le comprendre, le sujet de cet article restant JavaScript !

Les éléments qui nous intéressent sont les suivants :

- words est une fonction équivalente à split(' '), on l'utilise pour découper la ligne d'entrée (getLine correspond à readline)
- read est une fonction équivalente à parseInt
- comparing est une fonction qui permet de définir une logique de comparaison
- l'opérateur <> signifie « enchaîner deux comparaisons par ordre de priorité »
- minimumBy renvoie le minimum d'une liste selon une logique de comparaison

Ce qui est intéressant, c'est la correspondance de cette solution à notre todo-list : - on lit une chaîne de caractère et on la sépare en mots - on convertit chacun de ces mots en un entier - on définit une notion de distance à zéro - celle-ci se base sur la valeur absolue - et elle départage les ex-aequo par une comparaison "retournée" - on prend le minimum de la liste selon cette logique - et on l'affiche.

On est ici à une sorte de nirvana de la programmation dans lequel **énoncer clairement le problème et formuler sa solution sont un seul et même mouvement**. Si mes règles métier changent et que je veux changer la logique de comparaison (par exemple en prenant l'entier négatif plutôt que le positif pour départager deux températures de grandeur égale) j'ai un seul endroit du code à modifier. Mon code est **modulaire et composable**.

Un peu de cuisine fusion : les ingrédients de JS, le goût du fonctionnel

C'est bien joli, me direz-vous, mais moi je fais du JS et pas du Haskell ! Pas de panique, je vais quand même vous rassurer sur votre choix de langage :-) JS s'avère être un langage fonctionnel tout à fait idoine. Nous allons démystifier la soi-disant magie des langages fonctionnels en la découpant en petits éléments et en constatant que ces éléments sont bien présents dans JS !

Le premier bout de cette magie, peut-être l'un des plus importants, c'est la possibilité de **manipuler des valeurs de type fonction**, et de fournir ces valeurs à d'autres fonctions, qui peuvent décider de la façon dont elles vont les appeler. On parle de « higher order function » ou fonction d'ordre supérieur, et l'une de celles que l'on rencontre le plus souvent est la fameuse fonction map. (Peut-être en tirant un peu la métaphore culinaire, pourrions-nous les appeler des fonctions cannibales : des fonctions qui mangent d'autres fonctions…).

Pour rapprocher votre code JavaScript du nirvana fonctionnel, une heuristique s'impose parmi toutes : **cherchez à remplacer la plupart de vos boucles for par un appel à map (ou à l'une de ses cousines cannibales, comme filter)**.

Voici comment nous nous inspirons de cet adage pour améliorer le début de notre solution à l'exercice :

```
const n = parseInt(readline()); // the number of temperatures to analyse
var inputs = readline().split(' ');
temps = inputs.map(x => parseInt(x))
```

Nous utilisons la méthode map, native à la norme ES6/ES2015 et définie sur la classe Array, pour convertir une

liste de chaînes en une liste d'entiers. Cette méthode prend en argument une fonction, ici définie avec la syntaxe des "arrow functions", terme qui est approximativement synonyme de celui de "lambda". Et en l'occurrence cette fonction est celle qui convertit une chaîne (de caractères numériques) en un entier.

En sortant cette opération de la boucle for, nous avons déjà obtenu quelque chose d'utile : la séparation des préoccupations entre d'une part la représentation formelle des données de notre problème (le fait que les entrées du problème sont des entiers et que ces entiers sont encodés dans des chaînes de caractère), et d'autre part la sémantique du problème, ou si vous préférez son sens métier - le fait qu'il s'agit de grandeurs que nous cherchons à comparer.

Comme un petit goût de typage

Au passage, remarquons que la fonction map nous simplifie aussi la vie pour réfléchir à la « forme » de nos données. JavaScript n'est pas un langage typé, aussi c'est à nous et à notre cerveau aux capacités limitées qu'incombe la lourde responsabilité de vérifier la cohérence entre les parties de notre programme.

L'avantage d'utiliser des boucles explicites, c'est qu'on peut faire ce qu'on veut. L'inconvénient d'utiliser des boucles explicites… c'est qu'on peut faire ce qu'on veut, y compris des choses tellement complexes qu'on ne va plus les comprendre quand il s'agira de relire ou modifier le code que nous avons écrit.

La notion de « forme » est donc un outil pour se simplifier la vie, au prix de ne pas pouvoir faire tout et n'importe quoi.

Ainsi :

- nous avons dans l'exemple ci-dessus une forme de tableau ([]) de chaînes (string) que nous pouvons représenter comme : string[]
- la fonction parseInt, utilisée avec un seul argument, transforme une chaîne (string) en entier (number), ce qu'on peut représenter comme : string => number.
- en combinant ces deux formes avec map, nous allons donc nécessairement transformer un tableau de chaînes en tableau d'entiers string[] => number[]

Utiliser map garantit un certain nombre de propriétés du tableau résultat : il a le même nombre d'éléments que le tableau de départ, un tableau vide en entrée donnera un tableau vide en sortie. Schématiquement, map **ne change pas la forme du tableau, mais transforme tous les éléments qu'il contient**. Comme toutes ces propriétés font partie du « contrat » de map, je n'ai pas besoin de m'en préoccuper. Par exemple, ce n'est pas la peine d'écrire des tests unitaires pour les vérifier !

Ces propriétés sont très générales : si j'ai deux types foo et bar, que j'ai un foo[] et une fonction foo => bar , je peux invoquer map pour obtenir un bar[].

La « forme » de map est donc (foo[], (foo => bar)) => bar[]. Ça peut sembler abscons, car c'est écrit de façon très compacte, mais ça ne fait que résumer ce qu'on vient de dire !

Voici quelques autres exemples d'utilisation de map, qui vont également nous servir à introduire la syntaxe des « fonctions cannibales ».

```
temps = [-1, 3, -5, 4];

// syntaxe de lambda ou 'arrow function' anonyme
temps.map(x => Math.abs(x)); // [1, 3, 5, 4]
temps.map(x => x.toString()); // ["-1", "3", "-5", "4"]

// syntaxe de fonction 'classique'
function buzz(x) {if (x % 5 == 0) return "buzz"; else return "."};
temps.map(buzz); // [".", ".", "buzz", "."]

// syntaxe de lambda affectée à une variable
const fizz = (x) => {if (x % 3 == 0) return "fizz"; else return "."};
temps.map(fizz); // [".", "fizz", ".", "."]
```

Quant à la cousine filter (dont nous n'avons pas besoin pour cet exercice, mais qui est aussi très utile), elle prend une fonction foo => boolean et renvoie seulement les membres du tableau pour lesquelles elle est vraie, ses propriétés sont inverses : **filter change la forme du tableau, mais ne touche pas aux éléments qu'il contient**.

Si ce qui précède vous semble pétri de bon sens, alors réjouissez-vous, vous avez tout ce qu'il vous faut pour passer de JavaScript à TypeScript dont j'ai utilisé la notation pour vous parler de « forme ».

Réduire à petit feu…

Un troisième ingrédient de notre cuisine fonctionnelle est la méthode reduce. Elle est plus générale que map ou filter. Comme ces deux dernières, nous allons lui fournir une valeur de type fonction, qui est généralement appelée « fonction de réduction ».

La forme finale de la fonction obtenue est relativement simple : foo[] => bar. Nous partons d'un tableau d'éléments d'un type et nous obtenons un seul élément d'un autre type. On a donc réduit le tableau à une seule valeur finale.

La forme de la fonction de réduction est celle-ci : (foo, bar) => bar. Ce qui traduit bien la règle du jeu: une réduction ne peut s'opérer qu'en regardant un élément du tableau à la fois.

Quelle est la « forme » de reduce ? Si vous appliquez la même logique que pour map, vous allez trouver que c'est (foo[], (foo, bar) => bar)) => bar. C'est *presque* ça, mais pour tenir compte du cas d'un tableau vide il faudrait compliquer un peu plus.

Trouver l'élément minimum d'un tableau est une opération qui se prête bien à reduce, puisqu'on peut le faire en suivant cette règle du jeu.

On va toujours partir d'une valeur initiale Number.MAX_VALUE qu'on va comparer successivement à chaque élément du tableau. Évidemment, le premier élément du tableau est plus petit donc c'est celui-ci qu'on va garder. S'il en reste, on prend un autre élément du tableau, on le compare à celui qui est le plus petit jusqu'ici, et on garde le plus petit des deux, et ainsi de suite.

C'est exactement ce que fait notre boucle explicite dans la première version de la solution ! On peut donc transformer notre code pour utiliser reduce et en mettant de côté pour l'instant le cas du tableau vide. Voilà ce que ça donne :

Rappel de la première solution :

```
var closest = Number.MAX_VALUE;
if (n == 0) {
  closest = 0;
}
for (let i = 0; i < n; i++) {
  const t = parseInt(inputs[i]);// a temperature expressed as an integer ranging from -273 to 5526
  if (Math.abs(t) <= Math.abs(closest)) {
    if (Math.abs(t) != Math.abs(closest) || t > closest) {
      closest = t;
    }
  }
}
console.log(closest);

Nouvelle proposition:

function closeToZero(t, closest) {
  if (Math.abs(t) <= Math.abs(closest)) {
    if (Math.abs(t) != Math.abs(closest) || t > closest) {
      return t;
    }
  }
  return closest;
}

closest = temps.reduce(closeToZero, Number.MAX_VALUE)
```

Une épidémie de cannibalisme ?

Cette solution est déjà plus lisible : on a cantonné la complexité à la fonction closeToZero. Grâce à map et reduce, notre code est plus lisible et plus modulaire, donc plus fiable. On pourrait s'arrêter là, et si votre appétit est comblé, vous avez l'autorisation de sortir de table… mais je vous propose tout de même un petit dessert.

En effet, une partie de la complexité est toujours là, sous la forme de ces if imbriqués. Cette imbrication nous cache une idée importante, à savoir que nous raisonnons en deux étapes :

- d'abord je compare des valeurs absolues : par exemple entre -5 et 1, c'est 1 qui est plus près de zéro parce que 1 est inférieur à 5.
- ensuite s'il y a égalité, par exemple entre -5 et 5, c'est 5 que je vais considérer comme le plus proche de zéro donc « le plus petit » au sens de notre règle métier.

(La deuxième comparaison va donc dans le sens inverse, le plus petit est le plus grand et vice-versa; au lieu de comparer x on va comparer -x. C'est le principal piège de cet exercice !)

Dans l'exercice, on nous demande la température la plus proche de zéro, mais on pourrait aussi nous demander de les trier. On aurait probablement envie d'utiliser Array.sort(). Cette méthode (native ES6) effectue un simple tri arithmétique quand on l'emploie sans arguments, mais en réalité c'est une fonction « cannibale » : on peut aussi lui passer une fonction de comparaison en argument.

Une fonction de comparaison est de la forme (foo, foo) => enum {-1,0,1}. Elle prend deux valeurs du même type et elle renvoie -1 si la première est inférieure à la seconde, 0 s'il y a égalité, et 1 si la seconde est inférieure à la première.

Avec une fonction de comparaison, il est très facile d'implémenter la recherche d'un minimum en utilisant reduce, de façon générique.

```
function minBy(arr, compare) {
  function min (a, b) {
    if (compare(a,b) == -1)
      return a;
    else
      return b;
  }
  return arr.reduce(min);
}
```

On peut aussi préférer la syntaxe des lambda et l'opérateur ternaire pour une écriture plus compacte :

```
const minBy = (arr, compare) => arr.reduce((a,b) => compare(a,b) == -1 ? a : b)
```

La fonction de comparaison la plus simple est celle qui… compare deux valeurs.

```
temps = [-1, 3, -5, 4];
function compare (a,b) {return a < b ? -1 : (a == b ? 0 : 1)}
minBy(temps, compare) // -5
```

Oui, mais ce n'est pas très utile… par exemple, pour notre exercice, nous souhaitons comparer des valeurs absolues. Heureusement, rien ne nous empêche, maintenant que nous avons goûté à map et reduce, d'inventer nos propres recettes, c'est-à-dire créer nos propres fonctions cannibales.

```
temps = [-1, -5, 4, 3];
const comparing = (fn) => (a,b) => {fa=fn(a); fb=fn(b); return fa < fb ? -1 : (fa == fb ? 0 : 1)}
minBy(temps, comparing(Math.abs)); // -1
// tri par valeur absolue
temps.sort(comparing(Math.abs)); // [-1, 3, 4, -5]
// tri du plus grand au plus petit
temps.sort(comparing(x => -x)); // [4, 3, -1, -5]
```

Comme vous le voyez, rien de plus facile que de réutiliser une fonction de comparaison pour deux objectifs différents : trier le tableau, ou seulement trouver le plus petit.

Un zeste de monoïdes

Nous sommes presque au bout de nos peines, puisqu'il ne nous reste plus qu'à « enchaîner » deux comparaisons. En réalité, la logique d'une double comparaison était déjà présente en filigrane dans notre première version de closeToZero - effectuer la première comparaison, puis la deuxième seulement s'il y a égalité.

```
const chain = (cmp1, cmp2) => (a, b) => {r1 = cmp1(a,b); return r1 != 0 ? r1 : cmp2(a,b)}
```

En combinant ce qu'on a appris précédemment, on peut donc écrire :

```
const closeToZero = chain(comparing(Math.abs), comparing(x => -x))
```

Si vous lisez ce code à voix haute (et en anglais), vous vous apercevrez que vous lisez quelque chose de très proche de la « règle métier », à savoir « on enchaîne deux comparaisons, la première sur les valeurs absolues et la suivante sur les valeurs inversées ».

Cette idée d'enchaîner ou composer deux valeurs porte un nom en programmation fonctionnelle, on appelle ça un « monoïde ». En fait, c'est une idée qu'on retrouve partout : on peut cumuler deux chaînes (ça s'appelle concaténer), deux entiers (ça s'appelle additionner), deux booléens (ça s'appelle une porte logique)...

Ce qui est (légèrement) exotique à un palais habitué aux paradigmes impératifs ou objet, et spécifique à la programmation fonctionnelle, c'est l'idée qu'on peut **composer des valeurs de type fonction**.

De la même façon que vous n'avez pas besoin de compter sur vos doigts pour additionner 123 et 245, vous n'avez pas besoin d'utiliser deux if imbriqués pour exprimer la notion d'une double comparaison… et effectuer cette opération au bon niveau d'abstraction vous fait gagner du temps (compter sur ses doigts c'est très lent) et de la sécurité (moins de risque de sauter un chiffre et de devoir tout recommencer)… donc de la fiabilité.

J'ai dîné chez les fous ?

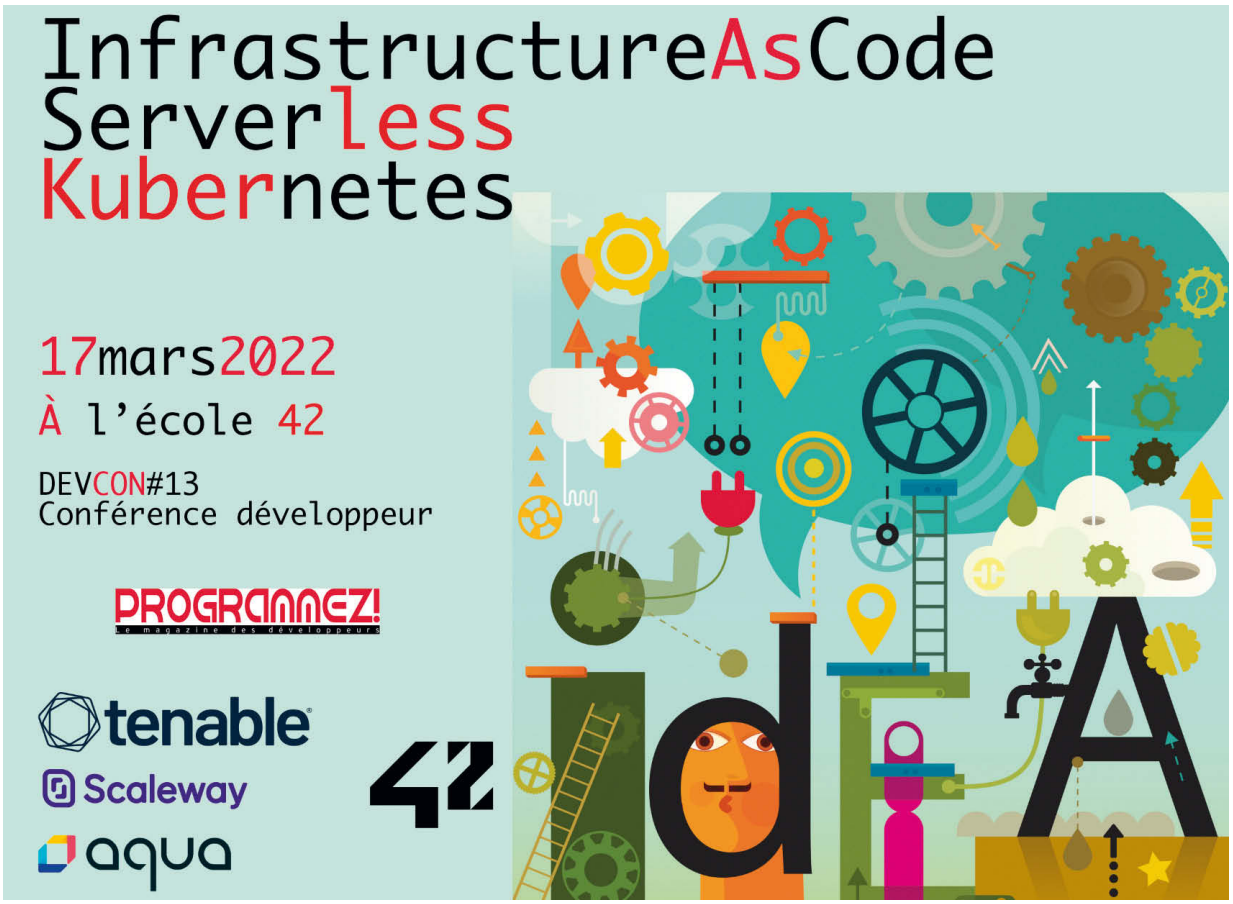
Voilà donc la solution à laquelle nous aboutissons :

```
parseInt(readline()); // on ignore la première ligne
var inputs = readline().split(' ');
temps = inputs.map(x => parseInt(x))

// Un trio de cannibales :)
const chain = (cmp1, cmp2) => (a, b) => {r1 = cmp1(a,b); return r1 != 0 ? r1 : cmp2(a,b)}
const comparing = (fn) => (a,b) => {fa=fn(a); fb=fn(b); return fa < fb ? -1 : (fa == fb ? 0 : 1)}
const minBy = (arr, order) => arr.reduce((a,b) => order(a,b) == -1 ? a : b)

const closeToZero = chain(comparing(Math.abs), comparing(x => -x))

closest = (n == 0) ? 0 : minBy(temps, closeToZero);
console.log(closest);
```

Dans une cuisine bien ordonnée, les fonctions chain, comparing et minBy font partie de votre équipement standard (c'est le cas en Haskell). Ce ne sont pas des ingrédients de la recette, donc elles ne comptent pas dans le nombre de lignes de la solution. Si on les supprime, on aura presque exactement la solution en Haskell que vous avez vue au début !

Nous allons donc nous quitter ici. Cette invitation à déguster un bon repas étant partie dans une direction tout à fait inattendue avec l'évocation des fonctions cannibales… on ne serait pas à l'abri que ça devienne une histoire de zombies ou quelque chose d'encore plus décalé ! Mais c'est aussi ça la magie du code, d'être un espace d'invention et de créativité dont n'est jamais exclu un grain de folie !

Si vous appréciez cet esprit, je vous conseille fortement la lecture de la série « Fantas, Eel, and Specification » de Tom Harding, qui m'a beaucoup inspiré pour cet article : http://www.tomharding.me/fantasy-land/



Écoconception d'un site web

Le développement durable est défini par le rapport Brundtland de la Commission Européenne en 1987 de la manière suivante : "Un développement qui répond aux besoins des générations présentes sans compromettre la capacité des générations futures à répondre aux leurs". Cette définition, donnée il y a plus de trente ans, est plus que jamais d'actualité. Il est urgent d'agir, si nous souhaitons continuer à profiter des bienfaits que nous apporte notre planète sans que cela remette en cause la qualité de vie des générations futures.

Hervé BOISGONTIER

Formateur en développement informatique au sein d'ENI École Informatique à Nantes.

Depuis près de 10 ans, avec différents langages, je forme à la programmation et l'art du développement informatique. Depuis longtemps, je m'intéresse au développement durable et depuis quelques années, comment le mettre en pratique dans notre métier d'informaticien ou d'informaticienne.

J'ai publié plusieurs livres aux éditions ENI dont le dernier : « Green IT et accessibilité – Développez votre site web Numérique Responsable ».

Souvent, nous avons l'impression que nous n'avons pas le pouvoir de changer les choses, car nous ne sommes ni dirigeants d'un pays ni PDG d'une grande entreprise. Néanmoins, ces dernières années ont montré que lorsque les citoyens du monde changent de regard, changent leurs habitudes, prennent conscience des enjeux, alors c'est toute notre société qui évolue. À nous, donc, de changer nos pratiques, dans notre vie de tous les jours, mais également dans notre vie de développeur !

Le green IT

Le numérique semble à tort ne pas avoir de conséquences écologiques, car il semble immatériel. Il nécessite néanmoins des équipements informatiques à la fois chez les utilisateurs (ordinateurs, tablettes, smartphones et box internet), au niveau des réseaux informatiques (switches, routeurs, antennes…) ainsi que dans les datacenters (serveurs, climatiseurs, onduleurs…). Tous ces équipements ont une empreinte écologique due à leur utilisation, mais également au niveau de leur production (la production représente les ¾ des impacts environnementaux globaux). Les impacts écologiques sont de plusieurs ordres : la consommation d'électricité, l'utilisation de minerais rares, la pollution de grande quantité d'eau, mais également du sol et de l'air, la production de nombreux déchets… Cela a entre autres pour conséquence une production de 4% des gaz à effets de serre et cela croit d'environ 6 à 9% par an ! Ces chiffres proviennent de l'Adem et du Shift Project. N'hésitez pas à consulter leurs rapports pour plus d'informations.

Le green for IT

Cette démarche vise à rendre plus vert, plus écologique le monde numérique. Il y a différentes facettes qui permettent cela :

- Sensibiliser le public aux conséquences de leurs actes numériques
- Revoir nos usages, nos pratiques
- Repenser notre besoin en équipements informatiques
- Faire durer le plus possible ces équipements
- Écoconcevoir nos services numériques
- Faire des choix responsables pour notre hébergeur, notre fournisseur d'électricité…

Dans cet article, le focus est fait sur la partie écoconception web.

L'IT for Green

Le numérique est également une formidable opportunité pour réduire notre empreinte écologique. Il peut permettre d'éviter des déplacements. Par exemple, une plateforme de covoiturage permet d'éviter que des voitures ne circulent pour une seule personne. Une solution de vidéoconférence est un autre exemple permettant d'éviter un certain nombre de déplacements. Le numérique peut également éviter du gaspillage. Par exemple avec une solution permettant une meilleure gestion des stocks ou d'optimiser l'utilisation de ressources…

La conception du service numérique

Cette étape est cruciale pour créer un service numérique avec une faible empreinte environnementale. Les plus gros leviers de gain green IT sont accessibles durant cette étape. Par la suite, il est possible d'effectuer des améliorations, mais celles-ci auront des impacts beaucoup plus faibles.

La conception d'un service numérique peut aboutir à des solutions utilisant peu de technologie ou des technologies anciennes et avec un faible impact écologique. Ces solutions se nomment low-tech en opposition à la high-tech. Cela peut par exemple avoir un service numérique consultable même sur un matériel ancien ou avec un réseau de faible débit…

Il est également possible d'utiliser des solutions n'utilisant pas du tout de technologie, c'est la no-tech. Par exemple, l'utilisation de chiens pour détecter précocement à l'odeur certains cancers permet d'obtenir de meilleurs résultats qu'avec une intelligence artificielle.

Bien souvent, les meilleures solutions allient plusieurs niveaux technologiques parmi les trois (no-tech, low-tech et high-tech). Il existe au Rwanda et au Ghana des livraisons de sang, médicaments et de vaccins par drone vers des communautés éloignées. Les commandes sont passées par SMS, ce qui est une bonne solution puisque ces pays n'ont pas une couverture suffisante pour utiliser internet. La solution alliant high-tech (drone) et low-tech (SMS) est une solution très écologique puisqu'elle a une empreinte écologique bien moindre que l'acheminement par voie terrestre et permet de sauver beaucoup de vies grâce à sa bien meilleure rapidité d'acheminement.

Lorsque le service numérique est repensé de fond en comble pour trouver une solution novatrice et à fort impact écologique, cela se nomme une action de facteur 4 par opposition

à celles ayant un impact plus faible nommé facteur 1.
Pour illustrer cela de manière imagée, prenons l'exemple de la réalisation d'un déplacement avec un véhicule. La solution de la voiture thermique correspond à une solution qui n'a pas été écoconçue. La solution d'une voiture électrique est une solution de facteur 1 : elle n'a pas remis en question l'utilisation de la voiture, mais à chercher à optimiser écologiquement parlant cette solution. Enfin, l'utilisation d'un vélo est une solution de facteur 4 : l'impact est réduit drastiquement.
Dans nos choix, il faut se méfier des reports. Cela est peut se produire lorsqu'une solution permet de réduire l'empreinte environnementale directe, mais augmente son empreinte indirecte. C'est par exemple le cas d'un véhicule électrique. À l'usage, son empreinte est bien moindre par rapport à un véhicule thermique, mais il faut créer et recycler les batteries de ces véhicules, ce qui est plus polluant que la création et le recyclage d'un réservoir de carburant.
Enfin, une solution peut être globalement meilleure, mais cela décuple son utilisation et au final a une empreinte écologique plus importante. C'est par exemple ce qui s'est passé avec la mise en place de la 5G en Corée du Sud.

Les fonctionnalités utiles

Alizée Colin dans un article de son blog lebondigital.com indique « un site est fini lorsqu'il n'y a plus rien à enlever et non à ajouter ». Cette phrase surprenante résume très bien la qualité première attendue pour un site internet écoconçu : la frugalité. Nous avons tous tendance à vouloir enrichir nos sites internet de nouvelles fonctionnalités, de nouvelles pages. Le terme enrichir fait penser que le site s'améliore, mais en fait c'est comme des aliments trop riches : ils nous font grossir et nous retrouvons en surcharge pondérale voire obèse ! Il en va de même pour les logiciels et en particulier les sites internet, certaines personnes parlent d'obésiciel ou de gras numérique ! Il est donc nécessaire de commencer une cure d'amincissement pour nos sites existants et de faire attention au régime alimentaire de nos futures créations.
Près de la moitié (45%) des fonctionnalités d'une application ne sont jamais ou très rarement utilisées. Il est donc stratégiquement plus intéressant de se concentrer sur les fonctionnalités vraiment utiles. Moins de lignes de code à produire cela a pour conséquence plus de temps à consacrer à la qualité du code, à l'accessibilité, à la sécurité, au respect des règles du green IT…
Cette bonne pratique est difficile à mettre en œuvre, car il faut convaincre ses clients, sa hiérarchie et ses collègues et faire preuve de beaucoup de pédagogie ! Pourtant sur le papier c'est la plus simple à réaliser puisque justement, il y a moins de chose à créer. Néanmoins c'est sans l'ombre d'un doute, celle qui aura le plus gros impact sur l'empreinte numérique de votre site internet ! Comme quoi la paresse est une bien belle qualité !
Pour chaque page internet ou fonctionnalité, il est important de se poser la question de sa réelle utilité et une fois en place de consulter les statistiques d'utilisation pour vérifier de sa pertinence.
Il est préférable d'adopter une approche "mobile first", c'est-à-dire en concevant le site en premier lieu pour les smartphones avant d'envisager les plus grands écrans. La faible taille de l'écran, nous incite à nous concentrer sur l'essentiel.

Le design

Le design d'un site est très important pour le marketing, pour l'image de marque, pour donner envie au visiteur… Et les graphistes effectuent souvent un travail très esthétique et attrayant. Néanmoins, il faut bien être conscient que le style appliqué à une page doit être considéré comme un plus permettant d'embellir la page, mais ne doit en aucun cas être essentiel pour la consultation ou l'interaction avec la page.
Pour concevoir une interface élégante et à la fois légère, il est important que les graphistes sachent ce qui est techniquement possible de faire pour donner du style uniquement avec du CSS sans avoir recours à des images. S'il est possible de se passer d'images pour styliser une bordure par exemple, cela permet d'économiser des ressources.
Il est possible par exemple de donner une ombre à une zone en utilisant la propriété CSS `box-shadow`. Les bordures peuvent être agrémentées de bords arrondis avec la propriété `border-radius`. Une image de fond peut avantageusement être remplacée par un fond en dégradé de couleur. Les différentes valeurs de la propriété `background` permettent de créer des effets très variés : `linear-gradient(…)`, `radial-gradient(…)` et `conic-gradient(…)`.
D'autres effets sont prévus lorsque les CSS4 seront implémentées par les navigateurs…

Les images

Les images sont un élément fondamental pour un site internet. Elles permettent de lui donner une identité graphique et de le rendre attrayant. Néanmoins, il faut être attentif à deux choses : d'une part, le poids qu'elles représentent et d'autre part, l'information qu'elles apportent.
Concernant le poids, les images sont beaucoup plus volumineuses que le contenu textuel évoqué jusqu'à présent (HTML, CSS et JavaScript). Il va donc être nécessaire de choisir attentivement leur nombre (lesquelles sont réellement nécessaires ?), leur taille et leur format.
Concernant l'accessibilité, il faut être attentif à ce que l'image soit correctement retranscrite aux internautes utilisant un lecteur d'écran.

Le nombre d'images

Le premier réflexe à avoir pour un site écoconçu est l'évitement. Il faut donc se poser la question de l'apport d'une image pour le site. Si elle n'a pas une plus-value importante, il peut être judicieux de ne pas la mettre ! De plus, de nos jours, une charte graphique simple, épurée et avec seulement quelques images bien mises en valeur est beaucoup plus tendance qu'un site surchargé d'images.

Les formats d'images

Il existe une multitude de formats d'images supportés par le web. D'une part les formats matriciels (PNG, JPEG, GIF…) et de l'autre les formats vectoriels (SVG principalement). Les formats matriciels définissent la couleur de chaque pixel de l'image alors que les formats vectoriels définissent des formes

géométriques (rectangles, ellipses, segments de droite, courbes…).

Les logos, les icônes et les schémas

Le SVG (Scalable Vector Graphics) est un format vectoriel. L'avantage de ce format est que, quel que soit le facteur de zoom, l'image est toujours parfaite sans aucun effet de pixélisation. Ce format est donc bien adapté pour un logo, un schéma, un dessin ou une icône, mais il n'est pas adapté pour une photo puisque celle-ci peut difficilement être réduite à des formes géométriques à moins de souhaiter un effet artistique particulier ! Le SVG est un langage utilisant des balises comme le HTML. Il est donc possible de l'éditer à l'aide d'un éditeur de texte.

Exemple :


```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 184.25 184.25">
 <defs>
  <style>.ap{fill:#144a7f;}.pp{fill:#fff;}</style>
 </defs>
 <rect class="ap" width="184.25" height="184.25"/>
 <rect class="pp" x="36.26" y="154.91" width="112.14" height="3.68"/>
 <path class="pp" d="M96.53,124.4s1.89.33-.09,1.17a88.91,88.91,0,
0,1-30.12,5.26c-16.45,0-35.81-7.1-39.34-27.31-3.31-19,13.14-47.47,
52.14-63.24,0,0,30.75-12.43,56.26-3.78,0,0,23.26,7.08,20.7,31.78,0,0-.92,
21.9-26.85,35.7,0,0-10.95,6.33-10.49-.75l2.58-21.34s1.56-4.86-3.5-2.
39c0,0-2.3,1.3-1.84-.54,0,0,0-.28,1.2-.74l9-2.58s2.57-1,2,2.85l-2.49,20.
42s-1.11,4.75,2.21,3.31h0c10.57-3.31,18.39-16.09,18.39-16.09,11.4-19.
87,2.95-32.74,2.95-32.74-10.49-17.49-36.06-16.75-36.06-16.75-37.34,
0-59.59,22.27-59.59,22.27C24.38,84.48,33.57,106,33.57,106c12.69,32.
92,62.49,18.45,62.49,18.45Z"/>
 <path class="pp" d="M49,92.48c-.06-4.38,3.11-14.16,8.45-14.16,2.1,
0,2.5,1.69,2.5,3.37a15.18,15.18,0,0,1-.4,3.51Zm15.46,6.87c-2.49,2.16-5.
93,5-9.37,5-5.21,0-6.15-5.39-6.09-9.5,6.76-4.65,13.1-8.56,18.76-13.34a
12,12,0,0,0,.2-1.57c0-.19-.07-.39-.33-.39-.55,0,0,.33-2.43,1.81V80.14c0
-3.29-2.22-4-5.18-4C49.25,76.11,41.62,86,41.62,96.2c0,6.39,2.7,11.18,
9.72,11.18a19.13,19.13,0,0,0,14-6.6Z"/>
 <path class="pp" d="M74.86,79.29A32.87,32.87,0,0,1,86.4,75.87c.
74,0,1,.13,1,1.33,0,2.56-.68,5.78-1.15,11.46l.14,12c2.29-5.39,6.88-12.
91,12.87-12.91,3.42,0,4.43,3.42,4.43,6.24s-2,17.4-1.94,18.73c0,.47,0,2.
08.8,2.08s3.65-1.48,4.59-2l.54,1.55c-2.63,2.21-9,4.16-11.4,4.16-1.41,
0-1.41-2.69-1.41-3.49,0-4.43,2.35-15.09,2.38-20.26,0-1.15-.34-2.37-1.
83-2.37-2.69,0-5.05,3.93-6.07,5.84-3.37,6.13-4,9.7-5.39,19.46H77.43c1.
07-8,3.1-21,3.1-24.15,0-1.22,0-2.3-1-2.3s-3.1,1-4.18,1.49Z"/>
 <path class="pp" d="M122.05,68.21a5,5,0,0,1,4.64-4.72,2.76,2.76,0,0,
1,2.76,2.91c0,2.28-2,4.72-4.37,4.72A2.84,2.84,0,0,1,122.05,68.21Z"/>
</svg>
```

Ce fichier peut sembler complexe, mais il possible de le comprendre dans les grandes lignes. Tout d'abord, il contient une balise racine <svg>. Celle-ci contient une balise <defs> incluant une balise <style> définissant deux styles : ap qui est utilisé pour l'arrière-plan et pp pour le premier plan. La première balise <rect> définit un rectangle couvrant toute l'image et permet de colorier tout le fond avec la couleur bleu. La seconde balise <rect> permet de dessiner le rectangle blanc du bas du logo. Enfin les balises <path> définissent des chemins constitués de courbes (définie à l'aide de points de contrôle de courbes de Bézier), d'arcs de cercle et de segments de droite.

Cette image ne pèse que 1 959 octets soit moins de 2Ko. Une image au format PNG équivalente avec un format de 100×100 pixels pèse 4 093 octets soit plus de deux fois plus avec l'inconvénient d'obtenir une image pixélisée si elle est zoomée.

Il est possible de créer des images vectorielles avec des outils tels que Inkscape (sous licence GNU GPL), Adobe Illustrator (propriétaire) ou d'utiliser des convertisseurs en ligne.

Le format PNG (Portable Network Graphics) est un format d'image ouvert contrairement au GIF (Graphics Interchange Format) qui est propriétaire. Il utilise une compression sans perte et est bien adapté aux images ayant des grands aplats de couleurs unies (permettant une compression efficace). Même si la règle d'utiliser le format SVG pour les logos, schémas et icônes fonctionne plutôt bien, il se peut qu'une image PNG soit plus légère si l'image est très complexe et requière la définition d'un très grand nombre de chemins en SVG. Il ne faut donc pas hésiter à comparer.

Les fonts icônes sont une police de caractères où chaque caractère représente une icône à la place d'un caractère. Cela a l'avantage de ne nécessiter le téléchargement que d'un seul fichier, la font, pour l'ensemble des icônes du site. Les caractères d'une police sont toujours définis vectoriellement afin de pouvoir parfaitement s'afficher, quelle que soit la taille de la police.

Les images animées

Il y a quelques décennies, le GIF animé était la grande tendance des sites internet : le site était actif et donnait une image dynamique… De nos jours, c'est beaucoup moins à la mode et c'est une bonne chose ! D'une part, cela constitue un élément de distraction qui est perturbant pour la plupart des personnes, mais qui peut rendre le reste du contenu inaccessible pour des personnes ayant des troubles de l'attention par exemple. D'autre part, un GIF animé est plus consommateur de ressources qu'une courte vidéo équivalente, qu'un PNG animé (Animated Portable Network Graphics) ou qu'un SVG animé. Ce dernier est à privilégier, car en plus d'être très léger, il n'est pas toujours parfaitement lisse, quelle que soit la taille d'affichage.

Dans un SVG animé, les éléments graphiques qui doivent bouger ensemble doivent être regroupés à l'aide d'une balise <g>. Au sein de celle-ci une balise <animateTransform> est ajoutée. Celle-ci est paramétrée avec différents attributs. L'attribut type pour indiquer le type de mouvement :

Type de mouvement	Valeur de l'attribut
Translation	translate
Rotation	rotate
Mise à l'échelle	scale
Déformement horizontal	skewX
Déformement vertical	skewY

Les attributs begin et dur définissent respectivement le moment du commencement du mouvement et sa durée. Ces deux valeurs sont indiquées en secondes avec un s derrière. Exemple : 2s pour deux secondes.
Les attributs from et to permettent de définir la position initiale et la position finale pour la transformation.
Par exemple, en reprenant le logo de l'ENI, il est possible de l'animer en ajoutant une rotation.

```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 184.25 184.25">
 <defs>
  <style>.ap{fill:#144a7f;}.pp{fill:#fff;}</style>
 </defs>
 <rect class="ap" width="184.25" height="184.25"/>
 <rect class="pp" x="36.26" y="154.91" width="112.14" height="3.68"/>
 <g>
  <path class="pp" d="…"/>
  <path class="pp" d="…"/>
  <path class="pp" d="…"/>
  <path class="pp" d="…"/>
  <animateTransform attributeName="transform" attributeType="XML"
type="rotate" from="360 92 80" to="0 92 80" dur="2s" repeatCount="1"/>
 </g>
</svg>
```

Les photographies

Les appareils photo ont des capteurs de plus en plus performants et produisent donc des photographies avec une meilleure résolution. Le problème est que ces fichiers images sont également de plus en plus volumineux. Il est donc important de faire un choix quant au format utilisé, au taux de compression, à la taille et à la résolution pour ces photographies.
Le format JPEG utilise une compression avec perte ou sans perte. Il est donc nécessaire de choisir le bon compromis entre une image parfaite et un poids d'image faible. Néanmoins la perte de qualité de l'image due à sa compression permet d'obtenir pour une photo des fichiers moins volumineux qu'avec du PNG. L'avantage de ce format est qu'il est supporté parfaitement par tous les navigateurs.
Ces dernières années de nouveaux formats d'images ont fait leur apparition. Ils ont pour objectifs de proposer des alternatives au format JPEG en corrigeant ses défauts (artefacts visibles avec une forte compression et non support de la transparence).
Le format WebP est un format proposé par Google permettant une meilleure compression que le JPEG à qualité équivalente. Le gain est d'environ 30%. Le problème est que ce format n'est supporté que récemment par la plupart des navigateurs.
Le format AVIF est un format à usage gratuit proposé par une alliance de grosses entreprises (Microsoft, Google, Vimeo, Amazon, Netflix…). Il est encore plus performant que le WebP (gain de l'ordre de 20%), mais n'est à l'heure actuelle supporté que part les navigateurs Chrome, Opéra et Firefox.
Avec les dernières versions de GIMP, il est possible d'exporter une image dans le format WebP ou AVIF. Il est également possible d'utiliser un convertisseur en ligne ou installé sur votre machine.
Exemple avec une image de test :

Format	Poids	Gain par rapport au JPEG
JPEG	872 Ko	0%
WebP	479 Ko	45%
AVIF	218 Ko	75%

Il est possible de proposer plusieurs formats d'image afin de limiter l'utilisation du réseau et que n'importe quel navigateur puisse afficher l'image. Pour cela, la balise <picture> permet lister les fichiers à utiliser par ordre de préférence. Si le navigateur ne supporte pas le format alors il passe au suivant. À la fin, une balise <img> permet d'afficher une image si aucun des formats précédents n'est supporté. C'est sur cette balise <img> que doit être positionnée l'alternative textuelle à l'image et le type de chargement qui sont utilisés quel que soit le format d'image utilisé par le navigateur.

Syntaxe :

```
<picture>
 <source srcset="image 1" type="type MIME de l'image 1">
 <source srcset="image 2" type="type MIME de l'image 2">
…
 <img src="image n" alt="..." loading="eager|lazy">
</picture>
```

Exemple :

```
<picture>
 <source srcset="images/fleurs.avif" type="image/avif">
 <source srcset="images/fleurs.webp" type="image/webp">
 <img src="images/fleurs.jpg" alt="...">
</picture>
```

Seule l'image supportée par le navigateur est téléchargée. Dans cet exemple, Chrome télécharge l'image au format AVIF. Une ancienne version de Firefox télécharge l'image au format WebP et Internet Explorer télécharge l'image au format JPEG. Il est possible de vérifier cela en consultant l'onglet réseau des outils du développeur de chaque navigateur.
Le fait de stocker plusieurs images sur le serveur à la place d'une seule est écologiquement très vite rentabilisé, car il faut deux fois plus d'énergie pour transporter une donnée que pour la stocker pendant un an (source GreenIT.fr).
Pour utiliser différents formats pour une image de fond, cela est plus compliqué. Avec les CSS 4 (encore à l'état de brouillon), il devrait être possible d'utiliser également un ensemble d'images pour que le navigateur utilise celle qu'il est capable d'afficher. Firefox est le premier et le seul navigateur à le proposer pour le moment.

Exemple :

```
.fondImage {
 background-image: url('images/fleurs.jpg');
 background-image: image-set(
  url('images/fleurs.webp') type("image/webp"),
  url('images/fleurs.jpg') type("image/jpeg")
 );
 …
}
```

La première règle background-image est définie pour les navigateurs ne supportant pas encore image-set et la seconde règle background-image écrase celle-ci si le navigateur le supporte.

À l'heure actuelle, pour réaliser l'équivalent sur l'ensemble des navigateurs, il est nécessaire d'utiliser un code JavaScript afin d'ajouter des classes (webp ou notwebp, avif ou notavif) à la balise html selon le support de ces formats par le navigateur. Vous pouvez trouver par exemple sur github leechy/img-support qui réalise cela.

Dans le fichier HTML :

```
<script src="imgsupport/imgsupport.js"></script>
```

Il suffit ensuite d'écrire des règles CSS en tenant compte de ces classes :

```
.avif .fondImage {
  background-image: url('images/fleurs.avif');
}
.webp.notavif .fondImage {
  background-image: url('images/fleurs.webp');
}
.notwebP.notavif .fondImage {
  background-image: url('images/fleurs.jpg');
}
```

Dans cet exemple, un navigateur supportant le format AVIF prend en compte que la première règle CSS. Un navigateur supportant WebP, mais pas AVIF met en œuvre uniquement la seconde. Il est important de bien préciser dans ce cas .notavif, car sinon les deux premières règles sont appliquées et les deux images sont chargées ! Enfin les navigateurs ne supportant ni AVIF, ni WebP utilisent la dernière règle CSS.

La transparence

Un critère de choix important pour la sélection du format d'une image est le support de la transparence. Parmi les formats cités précédemment, seul le format JPEG ne gère pas la transparence. Les formats SVG, PNG, WebP et AVIF supportent sans problème la transparence.

La taille

La taille des écrans des visiteurs de nos sites web est extrêmement variable allant de l'écran d'un petit smartphone à celui d'une grande télévision. Il n'est pas nécessaire de faire charger des images immenses pour les afficher en tout petit sur un smartphone. Pour une image au format vectoriel, il n'y a pas de problème, mais pour les images au format matriciel, il est préférable d'utiliser différentes résolutions d'images. Cela permet un gain substantiel puisque toujours avec l'image de test précédente voici la taille des fichiers :

	1200×1600	600×800	300×400
JPEG	872 Ko	271 Ko	88 Ko
WebP	479 Ko	158 Ko	55 Ko
AVIF	218 Ko	84 Ko	33 Ko

En téléchargeant l'image à la bonne dimension, il est possible de diviser presque par dix son poids.
À nouveau, la balise <picture> vient à notre rescousse pour proposer différentes images en fonction de la taille de l'écran (ou plus précisément de la taille du viewport si le navigateur n'est pas en plein écran). Il est possible d'utiliser des media queries pour choisir l'image avec la résolution la mieux adaptée.

Syntaxe :

```
<picture>
 <source srcset="image 1" media="mediaquery pour l'image 1">
 <source srcset="image 2" media="mediaquery pour l'image 2">
…
 <img src="image n" alt="…">
</picture>
```

Exemple :

```
<picture>
 <source srcset="images/petite.jpg" media="(max-width: 619px)">
 <source srcset="images/moyenne.jpg" media="(min-width: 620px)
and (max-width: 1219px)">
 <img src="images/grande.jpg" alt="">
</picture>
```

Il est également possible de proposer uniquement des photos de petite taille, quelle que soit la taille de l'écran avec éventuellement un lien pour télécharger l'image avec une meilleure résolution.

La compression

Les formats JPEG, WebP et AVIF possèdent tous des algorithmes de compression avec ou sans perte. Il est possible régler le taux de la compression pour ajuster le compromis entre la qualité de l'image et son poids. Une image JPEG provenant d'un appareil photo ou d'un smartphone n'est généralement pas ou peu compressé. Il est donc possible d'obtenir un gain significatif en ajustant ce taux. Il est possible de comprimer assez fortement une image tout en conservant une image de bonne qualité pour un site internet (mais pas pour un tirage photo). D'une image à une autre, le taux de compression acceptable n'est pas le même. Par exemple, avec le format JPEG, une photographie avec un ciel dégradé ne peut pas subir un fort taux de compression sans que l'image subisse des artefacts visibles.
Finalement, il est souvent nécessaire de produire beaucoup d'images :

- Pour les formats supportés par les différents navigateurs ;
- Pour les tailles différentes ;
- Pour tester différents facteurs de compression.

Réaliser cela image par image avec un logiciel de retouche

d'images est très chronophage. Il est donc intéressant d'automatiser cela. Par exemple, il est possible d'utiliser un convertisseur tel que joedrago/colorist (sur github) et d'écrire un script permettant avec son aide de générer tous les fichiers souhaités.
Voici un exemple de script PowerShell permettant de convertir les images au format JPEG présentes dans le répertoire **imagesSources** en images aux formats JPEG, WebP et AVIF, en taille native, en 800×600 (en supposant que l'image à une résolution 4:3) et en 400×300 :

```
$fichiers = Get-ChildItem '.\imagesSources\*JPEG'
foreach ($fichierSource in $fichiers) {
 $baseNom = $fichierSource -replace '^(.*\\)imagesSources(\\.*)\.jpg$',
'$1imagesCompressees$2'
 $info = colorist identify $fichierSource --json
 $w = $info -replace '^.*"width":(\d+).*$', '$1'
 $h = $info -replace '^.*"height":(\d+).*$', '$1'
 if ($w -lt $h) { #portrait
  $largeur = 600
 } else { #paysage
  $largeur = 800
 }
 #Grandes
 colorist convert $fichierSource $baseNom'-Grande.jpg' -q 45
 colorist convert $fichierSource $baseNom'-Grande.webp' -q 60
 colorist convert $fichierSource $baseNom'-Grande.avif' -q 55 --speed 0
 #Moyennes
 colorist convert $fichierSource $baseNom'-Moyenne.jpg' -q 45 --resize
$largeur
 colorist convert $fichierSource $baseNom'-Moyenne.webp' -q 60 --
resize $largeur
 colorist convert $fichierSource $baseNom'-Moyenne.avif' -q 35 --speed 0
 --resize $largeur
 #Petites
 $largeur /=2
 colorist convert $fichierSource $baseNom'-Petite.jpg' -q 35 --resize
$largeur
 colorist convert $fichierSource $baseNom'-Petite.webp' -q 45 --resize
$largeur
 colorist convert $fichierSource $baseNom'-Petite.avif' -q 25 --speed 0
 --resize $largeur
}
```

Ce fichier doit être enregistré avec l'extension .ps1 dans un dossier contenant le dossier **imagesSources** avec les images à convertir et un dossier **imagesCompressees** initialement vide. Bien évidemment ce script est à adapter en fonction de vos besoins (taille des images, format, qualité…) !

Le cache

Le cache est un très bon moyen pour optimiser son site internet et ainsi réduire son empreinte environnementale. Il existe plusieurs niveaux de cache.

Le cache navigateur

Lorsqu'un visiteur navigue pour la première fois sur votre site, il charge tous les éléments pour afficher votre page internet (page HTML, feuilles de styles CSS, scripts Javascripts, images…). Tous ces éléments sont conservés en cache un certain temps. Ainsi, s'il navigue sur une autre page du site, il n'a pas besoin de tous les fichiers, mais seulement de ceux qu'il ne possède pas. Si vous avez le logo de votre entreprise sur toutes vos pages, une feuille de style commune… ces éléments ne sont pas téléchargés à nouveau. Si l'internaute revient sur une page déjà visitée, normalement aucune ressource n'est envoyée depuis le serveur ce qui permet de le soulager et de moins solliciter le réseau.

Pour indiquer le temps pendant lequel le navigateur doit utiliser la ressource en cache plutôt que de la redemander au serveur, il est nécessaire de préciser dans l'en-tête HTTP Cache-Control: max-age=tempsEnSecondes, public. En configurant, Apache, il est possible de lui faire intégrer cette information dans ses réponses. Il faut le faire dans le fichier de configuration d'apache ou dans un fichier .htaccess à la racine de son site.

Syntaxe :

```
<filesMatch "filtreFichier">
  Header set Cache-Control "max-age=tempsEnSecondes, public"
</filesMatch>
```

La chaîne de caractère filtreFichier est une expression rationnelle PCRE.
Il est possible par exemple de conserver en cache les pages, les scripts et les feuilles de styles pendant dix jours (864 000 secondes) et les images pendant un an (31 536 000 secondes).

```
<filesMatch ".(css|js|html)$">
  Header set Cache-Control "max-age=864000, public"
</filesMatch>
<filesMatch ".(ico|webp|png|gif|svg|avif|jpg)$">
  Header set Cache-Control "max-age=31536000, public"
</filesMatch>
```

Pour vérifier le bon fonctionnement, il faut se rendre pour la première fois sur l'une des pages de notre site et visualiser la réponse à la requête HTTP envoyée dans les outils du développeur.

```
HTTP/1.1 200 OK
Date: Sun, 25 Jul 2021 12:58:29 GMT
Server: Apache/2.4.43 (Win64) OpenSSL/1.1.1g PHP/7.4.7
Last-Modified: Sun, 25 Jul 2021 12:58:27 GMT
ETag: "712-5c7f23156590a"
Accept-Ranges: bytes
Content-Length: 1810
Cache-Control: max-age=86400, public
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
```

Nous pouvons constater que l'en-tête contient bien la ligne Cache-Control.
Sinon actualisons la page, nous obtenons un code 304 pour indiquer que la page est celle récupérée en cache :

```
HTTP/1.1 304 Not Modified
Date: Sun, 25 Jul 2021 13:02:32 GMT
Server: Apache/2.4.43 (Win64) OpenSSL/1.1.1g PHP/7.4.7
Connection: Keep-Alive
Keep-Alive: timeout=5, max=100
ETag: "712-5c7f23156590a"
Cache-Control: max-age=86400, public
```

Lorsque la ressource demandée est en cache, mais que celle-ci a expirée. Une requête est envoyée au serveur pour savoir s'il est encore valable avec comme en-tête :

```
If-Modified-Since: dateDuFichierEnCache
```

Si celle-ci n'a pas changée et le serveur renvoi le code 304 afin que la ressource en cache soit utilisée. Sinon la nouvelle version de la ressource est envoyée avec un code 200.

Le cache du serveur web

Le serveur Apache peut conserver également le résultat de l'exécution de code PHP pour éviter de la régénérer à chaque fois. Cela peut être réalisé en configurant Apache avec les lignes suivantes :

```
CacheEnable mem répertoireDuCache
MCacheSize tailleDuCacheEnOctets
MCacheMaxObjectCount nombreDElementsEnCache
MCacheMinObjectSize tailleMinimale
MCacheMaxObjectSize tailleMaximale
```

Exemple :

```
CacheEnable mem /cache
MCacheSize 100000000
MCacheMaxObjectCount 1000
MCacheMinObjectSize 100
MCacheMaxObjectSize 1000000
```

Ainsi, si plusieurs internautes demandent successivement une même page générée dynamiquement en PHP, elle est générée pour être envoyée au premier et mise en cache sur le serveur et le second reçoit la page stockée dans le cache. Cette technique est particulièrement intéressante si toutes les pages sont générées dynamiquement à l'aide d'un modèle de page.

Le cache de PHP

La génération et l'envoi d'une page dynamique sont plus coûteux que l'envoi d'une page statique directement. C'est pourquoi pour accélérer le rendu des pages dynamiques, le code PHP qui a déjà été analysé et transformé en langage intermédiaire (l'Opcode) est sauvegardé en cache pour éviter de refaire ces étapes. Jusqu'à la version 5.4 de PHP, il était nécessaire d'effectuer des étapes de configuration pour le mettre en place. Depuis PHP 5.5, l'OpCache est installé et activé par défaut. Il n'y a donc rien à faire si vous utilisez une version récente de PHP.

Le cache du serveur de bases de données

Il existe également la possibilité d'utiliser un cache pour les requêtes fréquemment envoyées à votre serveur de base de données. Il faut alors utiliser un produit supplémentaire tel que la solution open source Redis. Si les autres niveaux de cache ont été correctement mis en place, ce cache n'est nécessaire que sur un site internet sollicitant beaucoup la base de données.

Conclusion

Écoconcevoir un site internet peut sembler une tâche ardue, mais j'espère vous avoir convaincu que cela n'est pas plus compliqué que cela et que petit à petit, les réflexes d'écoconception vous viendront naturellement à l'esprit lors de vos créations : Est-ce que cet élément est essentiel ? Est -ce qu'il peut être évité ? Est-ce qu'il peut être remplacé par une alternative moins gourmande ? Est-ce la meilleure implémentation possible ?

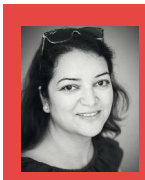
Dans cet article, seuls quelques aspects du numérique responsable ayant le plus d'impacts ont été abordés. Il y a bien d'autres aspects à prendre en compte. Tout d'abord, des choix techniques de développements ont des impacts significatifs (l'utilisation de frameworks et de bibliothèques, les médias audio et vidéo, les animations, les cartes, les traitements côté client et côté serveur, la gestion des données, les impressions…). Ensuite, le choix d'un hébergeur vert peut réduire l'empreinte écologique de votre site. Enfin, il faut apprendre à auditer un site pour réfléchir aux aspects à retravailler. Retrouvez l'intégralité de mon expertise sur le sujet de l'écoconception, mais également sur l'accessibilité à tout public de nos sites internet dans mon livre « Green IT et accessibilité - Développez votre site web Numérique Responsable » aux éditions ENI.

Toujours Disponible

HORS SÉRIE
100% JEUX

Quand et pourquoi refactorer (son code) ?

Un jour, un des développeurs de mon équipe a annoncé qu'il devait refactorer un bout de code, car il le trouvait sale. Je lui ai alors demandé plus de détails quant au type de refactoring. Était-ce à cause du nommage ou était-ce plus profond ?



Dorra Bartaguiz

Développeuse Azure/.net depuis plus de 13 ans, passionnée par le développement et les bonnes pratiques, accompagne les clients à travers du coaching ou des formations pour transmettre ses connaissances et aider les équipes en leur apportant de la valeur.

En creusant un peu, il s'est avéré qu'il avait en fait mis du temps à comprendre le code et cela n'était pas en accord avec sa façon de faire. Dans l'open-space, le chef d'équipe qui suivait la discussion, a rebondi en insistant « Il ne faut pas changer un code de production s'il n'y a pas de bug remonté ». Je n'étais et je ne suis toujours pas d'accord avec ce point de vue. Je considère que si un code est trop compliqué à appréhender, il doit être refactoré surtout s'il est bien couvert par des tests. Un refactoring est une modification ou un remaniement du code existant sans modifier le comportement. En théorie, le refactoring n'apporte rien à l'utilisateur final puisque le comportement ne change pas et c'est pour ça que plusieurs sont contre tout changement du code s'il n'y a pas de bug. De mon côté, je trouve qu'il y a plusieurs raisons de faire du refactoring :

- Quand un code est considéré comme sale
- Quand un code est incompréhensible
- Pour préparer une nouvelle fonctionnalité
- Planifier un refactoring non urgent pour plus tard
- Refactoring de longue haleine (plus long terme)

Refactorer un code sale

La définition d'un code sale est propre à chacun. Elle peut être aussi déterminée au niveau de l'équipe via des règles. Des règles de nommage, par exemple, peuvent permettre d'obtenir un code plus harmonieux. Prenons l'exemple ci-dessous, nous avons une méthode qui retourne un résultat à partir de deux entiers.

```
public double GetResult(int int1, int int2)
{
  if (int2 == 0)
  {
    throw new InvalidOperationException("Can't divide by zero !");
  }

  return (double)int1 / (double)int2;
}
```

À partir de la signature, on ne peut pas déterminer ce qu'elle fait. Il faut aller dans l'implémentation pour comprendre qu'il s'agit de diviser un numérateur par un dénominateur. Ce code peut donc être refactoré comme suit en nommant correctement les variables à minima.

```
public double Divide(int numerator, int denominator)
{
  if (denominator == 0)
  {
    throw new InvalidOperationException("Can't divide by zero !");
  }

  return (double)numerator / (double)denominator;
}
```

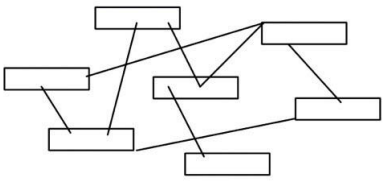
Prenons un autre exemple, la classe "File" a une propriété Name avec un setter publique.

```
public class File
{
  public File()
  {
  }

  public string Name { get; set; }
}
```

Certains ne trouveront pas ce code choquant et pourtant cette classe ne respecte pas l'immutabilité. Si dans l'équipe, l'immutabilité doit être respectée, il vaut mieux créer un champ privé initialisé via le constructeur et à minima avoir un getter pour accéder à la valeur, voire même supprimer la propriété pour rapprocher le comportement à la classe comme préconisé dans les règles des objets calisthenics ("No getters/setters properties") ou le principe de "Tell, don't ask". Le résultat peut ressembler à ce qui suit.

```
public class File
{
  private string name;
  public File(string name)
  {
    this.name = name;
  }

  public string Name
  {
    get
    {
      return this.name;
    }
  }
}
```

Refactorer un code incompréhensible



Parfois, je mets du temps à comprendre un bout de code. Je le lis, je le débugue pour essayer de comprendre. Je schématise les classes, les méthodes et le lien entre elles. A la fin, mon schéma ressemble à un sac de nœuds que je dois démêler.

Cette motivation est proche de la première, mais elle n'a pas la même intention. En effet, le collègue qui passera après mettra le même temps ou plus pour comprendre le code en question, si ce dernier ne change pas. C'est pourquoi je considère que ce refactoring est nécessaire pour gagner du temps pour délivrer les fonctionnalités futures. Pourquoi ne pas appliquer la règle des Boy Scouts qui se résume en une phrase : « Toujours laisser un endroit dans un meilleur état que celui où vous l'avez trouvé » ? »

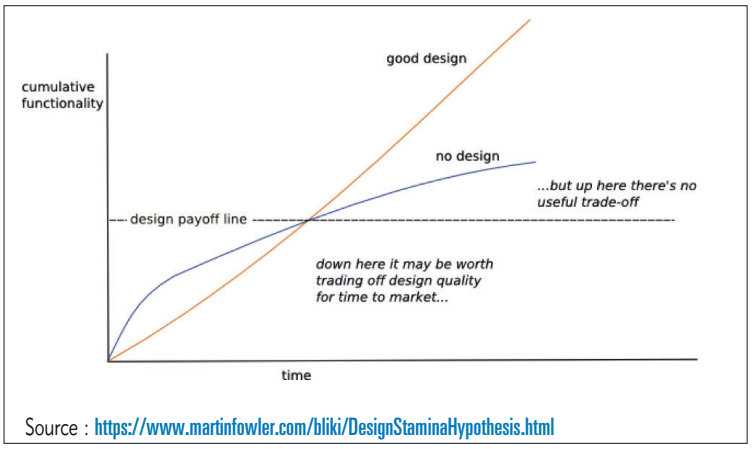


Figure 1 Source : https://www.martinfowler.com/bliki/DesignStaminaHypothesis.html

Refactorer pour préparer une nouvelle fonctionnalité

Vous devez rajouter un traitement dans une méthode existante, mais en l'état c'est compliqué à faire. Vous décidez donc de simplifier le code en extrayant une classe par exemple. Ce type de refactoring vous permet de gagner du temps de réalisation de la nouvelle feature.

Refactoring planifié

Vous êtes en train de rajouter du code pour la nouvelle feature, vous tombez sur du code à refactorer, mais vous n'avez pas le temps ou vous ne voyez pas l'intérêt de le faire à ce moment. Vous décidez donc de le noter sur un post-it, un ticket Jira ou un autre support pour ne pas l'oublier et pour le prioriser par la suite. C'est une bonne initiative car la livraison de la fonctionnalité ne sera pas impactée et en même temps, vous prendrez le temps pour le faire proprement juste après.

Refactoring de longue haleine (plus long terme)

Il s'agit d'un grand chantier de refactoring comme une nouvelle vision ou une nouvelle architecture de l'application. Vous allez par exemple passer d'un monolithe à une architecture micro-services. C'est un chantier qui risque de prendre du temps. Il vaut mieux donc déterminer les étapes à suivre avant de commencer directement à refactorer le code, voir les impacts sur l'application et si elles (les étapes) sont toujours compatibles avec de potentielles évolutions en attendant l'architecture cible.

Mais en vrai pourquoi refactorer ?

La réponse que j'entends souvent est "pour avoir un code de meilleure qualité" ou encore "pour avoir du code propre" ou même "pour faire les choses proprement". Au fil des expériences, j'ai vu que ces arguments ne valaient rien en face des clients, qui considèrent que ce type de refactoring est une perte de temps. A leurs yeux, l'application est en production, fonctionne correctement et n'a donc pas besoin de modification avec le risque d'y apporter des régressions. Pourtant je reste convaincue qu'une des facettes de notre mission en tant que développeu.r.se est de s'assurer que le code soit clean et s'améliore lorsqu'on intervient dessus. Ça fait partie de l'hygiène de vie de l'application.

Je représente souvent l'application comme un bébé. Ce dernier ne sait pas se laver tout seul. Il faut donc le surveiller, lui faire sa toilette si on trouve qu'il en a besoin. Une bonne hygiène lui permet de rester en bonne santé, d'éviter les infections et surtout d'avoir une bonne croissance. Pour une application c'est pareil. Il faut l'améliorer en continu avec du refactoring régulier couvert de tests bien sûr.

Si vous vous trouvez à court d'arguments, en voici un qui a le mérite d'exister. Il faut en faire pour faciliter l'ajout des fonctionnalités futures. L'amélioration continue du code permet de gagner du temps le jour où il faut passer sur le code en question pour y ajouter une nouvelle feature ou corriger un bug. Plus on refactore pour obtenir un bon design plus les fonctionnalités futures sont rapides à développer comme le montre le schéma de Martin Fowler sur le "Design Stamina Hypothesis" **Figure 1**

Mais quand s'arrêter ?

Faites attention à votre bébé (le logiciel). Il est fragile tout de même. Il ne faut pas le laver souvent non plus, car ça risque d'irriter sa peau surtout si l'eau est calcaire !

Un refactoring n'est pas une mince affaire. Il ne faut pas prendre le sujet à la légère et surtout il vaut mieux éviter de prendre la décision seul.e. En tant que développeu.r.se et surtout membre d'une équipe, votre devoir est d'informer les autres de vos intentions de refactoring pour deux raisons.

La première est que vous êtes dans une équipe et donc c'est important de communiquer et de partager. Il ne faut pas tomber dans le piège de la demande de permission non plus. Vous êtes majeur et responsable. Vous devez convaincre les autres des bienfaits de votre démarche.

Et la deuxième raison est pour éviter un éventuel problème de merge par exemple, surtout si vous travaillez en mode "feature branching" (gestion de branches de développement, une user story par branche et on merge une fois le développement fini pour cette user story).

C'est la responsabilité de chaque développeu.r.se de se demander si ce bout de code nécessite d'être refactoré. Posez-vous toujours ces questions "Est-ce que le code, tel qu'il est, me ralentit pour l'ajout d'une nouvelle feature ? Si je le modifie, est-ce que ça va me permettre de gagner du temps pour développer ma feature ou la correction de bug ?"

Si la réponse est oui alors informez l'équipe et faites le tout simplement. J'ai envie de dire que ça fait partie de la feature à développer.

Par contre si la réponse est non ou si vous n'êtes pas sûr, je vous conseille de ne pas le modifier et d'attendre le moment où cela sera vraiment nécessaire. Sinon faites-le dans le cadre d'un refactoring planifié.

Enfin, je vous invite à toujours garder ces phrases en tête : "Refactoring is about an economic argument. Clean code allows you to go faster. That's the only justification behind it." comme l'a annoncé Martin Fowler lors de sa conférence "Workflows of refactoring". Il ne s'agit donc pas d'un dogme, mais d'un raisonnement économique raisonnable !

J'espère que vous êtes mieux armés maintenant.

Si vous voulez vous exercer au refactoring, je vous recommande de jeter un coup d'œil au github d'Emily Bache par ici https://github.com/emilybache. Elle y recense plusieurs projets à refactorer avec plusieurs langages ;)

Ou alors vous pouvez vous exercer sur le kata Game Of Life sur mon github https://github.com/iAmDorra/GameOfLife. Plusieurs langages sont disponibles aussi grâce à l'effort collectif des amis que je remercie au passage :) A bientôt.

Deep Learning avec TensorFlow/Keras: contrôle de l'apprentissage avec les "callback"

Dans les n°244 & 246 du magazine, nous avons utilisé les outils TensorFlow/Keras [1][2] pour classer des images par reconnaissance de leurs contenus. La technique utilisée est celle du « deep learning » avec des réseaux de neurones capables « d'apprendre » à partir d'une base d'exemples. La bonne exécution de cette phase d'apprentissage est primordiale pour garantir de bonnes performances. Cet article détaille les mécanismes de « callback » qui permettent de contrôler très précisément le déroulement de l'étape d'entraînement.

Jean-Christophe Riat

Passionné depuis le lycée par l'informatique, j'enseigne l'intelligence artificielle depuis plus de 20 ans à l'EPSI Paris, 1re école d'informatique créée en France par des professionnels [7]. Je suis enthousiasmé par les progrès en apprentissage machine, d'autant qu'internet rend accessibles à tous les ressources (tutoriels, outils…) pour mettre en œuvre les techniques les plus récentes de « Deep Learning ».

Déroulement de la phase d'apprentissage

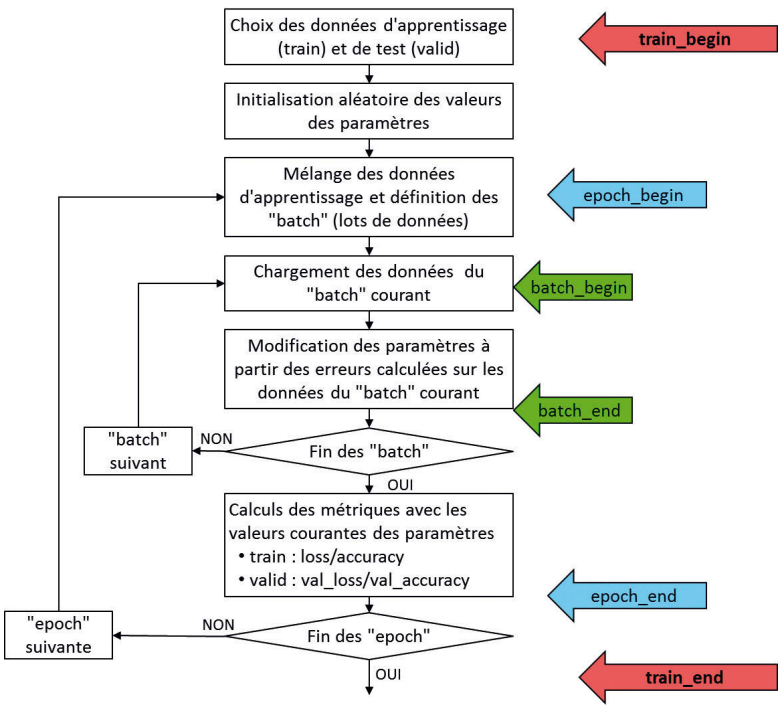
L'utilisation d'un réseau de neurones débute par la construction de son architecture, avec la définition des entrées et des sorties, l'organisation des couches de convolution/pooling et le choix du nombre et des dimensions des couches de classification. Chaque architecture comporte une multitude de paramètres (de quelques dizaines à plusieurs centaines de millions !), associés aux poids des connexions, aux biais des neurones et aux matrices de convolution. Le « deep learning » consiste à calculer des valeurs pour ces paramètres, à partir d'une base d'exemples représentatifs du comportement entrées/sorties souhaité. Généralement, une partie de cette base est réservée pour évaluer la capacité du réseau à généraliser ; les exemples ne sont alors pas utilisés pour l'apprentissage, mais uniquement pour comparer les sorties calculées à celles attendues.

Les algorithmes d'entraînement calculent les paramètres de manière itérative ; les valeurs sont initialisées aléatoirement, puis progressivement modifiées en exploitant les exemples de la base d'apprentissage. Comme illustré sur **l'encadré n°1**, ces calculs fonctionnent avec deux boucles imbriquées. La 1re boucle est une itération sur un nombre de cycles (epochs), avec à chaque fois l'exploitation de la totalité des données d'apprentissage. Celles-ci sont regroupées en lots (batch) de tailles identiques, par exemple 32, 64 ou 128 données. La 2de boucle permet le parcours des lots dans un cycle, avec à chaque itération un calcul pour modifier les valeurs des paramètres en fonction des erreurs de prédictions sur les données du lot. Le nombre de cycles (epochs) ainsi que la taille des lots (batch_size) sont des paramètres transmis dans les paramètres de la fonction d'apprentissage « *MonReseau*.**fit()** ».

Principes des mécanismes de « callback »

Une particularité des algorithmes d'apprentissage est d'être non déterministes, c'est-à-dire que l'exécution deux fois de suite du même algorithme sur les mêmes données ne produira pas les mêmes valeurs. Ceci est notamment lié à l'initialisation aléatoire des paramètres. Cette caractéristique peut surprendre, mais il n'existe pas, à ce jour, d'autres méthodes pour déterminer les paramètres d'un réseau de grande taille. Compte tenu des caractères progressif et non déterministe des algorithmes d'apprentissage, il est utile de pouvoir contrôler leurs déroulements. C'est dans ce but que la bibliothèque Keras intègre les mécanismes de « callback », disponibles dans le module « tf.keras.callbacks ». Une classe abstraite « Callback » est définie avec six méthodes (cf. **encadré n°2**) pour décrire les traitements à réaliser lors de certaines étapes de l'algorithme d'entraînement (cf. **encadré n°1**). Ces traitements sont à définir lors de la dérivation de la classe abstraite ; il n'est pas nécessaire de renseigner les six méthodes, mais uniquement celles qui correspondent aux étapes de l'algorithme dans lesquelles une opération est à réaliser (par

Encadré n°1: fonctionnement de l'algorithme d'apprentissage



Algorithme d'apprentissage avec 2 boucles imbriquées : une 1re boucle pour itérer sur le nombre de cycles (epoch) et une 2de boucle pour parcourir l'ensemble des lots de données (batch) à l'intérieur d'un cycle.

Encadré n°2: classe abstraite Callback

Déclaration de la classe abstraite

```
class Callback(object):
  """Abstract base class used to build new callbacks."""
```

Pour réaliser un contrôle en début et en fin de l'apprentissage

```
def on_train_begin(self, logs=None):
  """Called at the beginning of training."""

def on_train_end(self, logs=None):
  """Called at the end of training."""
```

Pour réaliser un contrôle en début et en fin d'un cycle d'apprentissage

```
def on_epoch_begin(self, epoch, logs=None):
  """Called at the start of an epoch."""

def on_epoch_end(self, epoch, logs=None):
  """Called at the end of an epoch."""
```

Pour réaliser un contrôle en début et en fin du traitement d'un lot de données

```
def on_train_batch_begin(self, batch, logs=None):
  """Called at the beginning of a training batch in `fit` methods."""

def on_train_batch_end(self, batch, logs=None):
  """Called at the end of a training batch in `fit` methods."""
```

Méthodes associées à la classe abstraite « Callback »: le code est à renseigner lors de la création des classes en fonction des traitements souhaités.

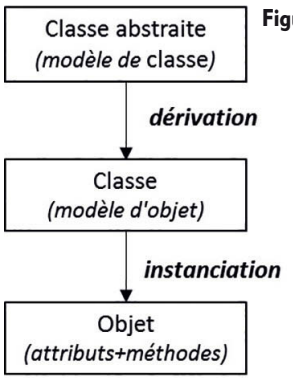


Figure 1

```
# Importation du module tensorFlow
import tensorflow as tf

# Chargement des données d'apprentissage et de tests
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Création d'un réseau multicouche
MonReseau = tf.keras.Sequential()
MonReseau.add(tf.keras.layers.Dense(units=50,
                                    input_shape=(784,),
                                    activation='relu'))
MonReseau.add(tf.keras.layers.Dense(units=10,
                                    activation='softmax'))

# COMPILATION du réseau
MonReseau.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

# APPRENTISSAGE du réseau
hist=MonReseau.fit(x=x_train,
                   y=y_train,
                   epochs=15,
                   verbose=2,
                   batch_size=32,
                   validation_data=(x_test,y_test),
                   callbacks=None)

# PERFORMANCES du réseau mémorisé
print("\nEvaluation sur la base de tests:")
perf=MonReseau.evaluate(x=x_test,y=y_test,
                        verbose=2)
print("Résultats du test: {:.2f}%".format(perf[1]*100))
```

Lancement de l'apprentissage

Figure 2

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
60000/60000 - 4s - loss: 0.3234 - accuracy: 0.9102 - val_loss: 0.1859 - val_accuracy: 0.9456
Epoch 2/15
60000/60000 - 3s - loss: 0.1627 - accuracy: 0.9532 - val_loss: 0.1546 - val_accuracy: 0.9545
Epoch 3/15
60000/60000 - 3s - loss: 0.1227 - accuracy: 0.9644 - val_loss: 0.1238 - val_accuracy: 0.9616
Epoch 4/15
60000/60000 - 3s - loss: 0.0985 - accuracy: 0.9706 - val_loss: 0.1067 - val_accuracy: 0.9662
Epoch 5/15
60000/60000 - 3s - loss: 0.0835 - accuracy: 0.9749 - val_loss: 0.1030 - val_accuracy: 0.9707
Epoch 6/15
60000/60000 - 4s - loss: 0.0716 - accuracy: 0.9785 - val_loss: 0.0987 - val_accuracy: 0.9710
Epoch 7/15
60000/60000 - 4s - loss: 0.0631 - accuracy: 0.9808 - val_loss: 0.0987 - val_accuracy: 0.9726
Epoch 8/15
60000/60000 - 3s - loss: 0.0544 - accuracy: 0.9838 - val_loss: 0.1013 - val_accuracy: 0.9695
Epoch 9/15
60000/60000 - 3s - loss: 0.0488 - accuracy: 0.9845 - val_loss: 0.0958 - val_accuracy: 0.9718
Epoch 10/15
60000/60000 - 4s - loss: 0.0434 - accuracy: 0.9865 - val_loss: 0.1024 - val_accuracy: 0.9717
Epoch 11/15
60000/60000 - 3s - loss: 0.0395 - accuracy: 0.9875 - val_loss: 0.1010 - val_accuracy: 0.9710
Epoch 12/15
60000/60000 - 3s - loss: 0.0341 - accuracy: 0.9894 - val_loss: 0.1068 - val_accuracy: 0.9702
Epoch 13/15
60000/60000 - 3s - loss: 0.0313 - accuracy: 0.9906 - val_loss: 0.1028 - val_accuracy: 0.9723
Epoch 14/15
60000/60000 - 3s - loss: 0.0302 - accuracy: 0.9905 - val_loss: 0.1002 - val_accuracy: 0.9712
Epoch 15/15
60000/60000 - 3s - loss: 0.0254 - accuracy: 0.9921 - val_loss: 0.1144 - val_accuracy: 0.9687

Évaluation sur la base de tests:
10000/10000 - 0s - loss: 0.1144 - accuracy: 0.9687
Résultats du test: 96.87%
```

Meilleur résultat différent de celui retenu

Figure 3

exemple « on_epoch_begin » pour le début d'un cycle ou « on_train_batch_end » pour la fin du traitement d'un lot). La dérivation de la classe abstraite permet de créer des classes puis des objets après instanciation : **figure 1.**

Lors d'un appel à la fonction d'entraînement « *MonReseau*.**fit()** », le paramètre « callback » est utilisé pour transmettre les objets associés aux traitements souhaités (cf. exemples ci-dessous).

Liste des classes prédéfinies

Pour simplifier l'utilisation des « callbacks », le module « tf.keras.callbacks » met à disposition des classes déjà définies, pour les cas suivants d'utilisation :

BaseLogger()	Sauvegarde des métriques de mesure de performance à la fin de chaque cycle (utilisé par défaut)
CSVLogger()	Sauvegarde des données d'apprentissage dans un fichier au format CSV à la fin de chaque cycle
CallbackList()	Regroupement de plusieurs callbacks
EarlyStopping()	Arrêt de l'apprentissage par contrôle de la progression de l'évolution sur une des métriques de mesure de performance
History()	Construction d'un objet « History » avec les valeurs des métriques de performance à chaque cycle (utilisé par défaut)
LambdaCallback()	Définition de traitements avec des lambda fonctions Python
LearningRateScheduler()	Définition d'une fonction pour modifier la valeur du taux d'apprentissage à chaque début de cycle
ModelCheckpoint()	Définition de critères pour sauvegarder le réseau durant l'apprentissage (régulièrement, sur une performance meilleure…)
ProgbarLogger()	Affichage de la progression et des valeurs des métriques sur le canal de sortie par défaut
ReduceLROnPlateau()	Modification du taux d'apprentissage sur détection d'un palier dans la diminution de l'erreur
RemoteMonitor()	Envoi en continu des informations de l'apprentissage sur une adresse de serveur
TensorBoard()	Exportation des données pour exploitation par l'outil TensorBoard
TerminateOnNaN()	Arrêt de l'apprentissage sur détection d'une valeur NaN sur la mesure de l'erreur

Exemples de mise en œuvre de callback prédéfinis

Pour illustrer l'utilisation des « callbacks », nous nous appuyons sur le perceptron multicouche de l'**encadré n°3** qui fonctionne

avec la base des chiffres manuscrits MNIST détaillée dans le n°244. Le programme Python associé est détaillé sur la **figure 2**. Un exemple de résultats à l'exécution est présenté sur la **figure 3.**

On observe que 'val_accuracy', qui est la métrique pour mesurer la performance en généralisation du réseau, ne progresse pas de manière constante, mais fluctue. En conséquence, en retenant les valeurs des paramètres à la fin des 15 cycles d'apprentissage, le réseau obtenu n'est pas nécessairement celui avec les meilleures performances !

Le callback « EarlyStopping » est une première solution d'amélioration. Il permet de suivre l'évolution de l'erreur 'val_accuracy' et d'arrêter l'apprentissage si aucun progrès n'est constaté pendant plusieurs cycles consécutifs. La partie « apprentissage » du programme précédent est à remplacer par le code de la **figure 4.**

Le paramètre « patience », dont la valeur est ici fixée à 3, définit le nombre de cycles à considérer pour surveiller l'évolution de la métrique. À la fin de l'apprentissage, les paramètres retenus sont ceux associés à la meilleure valeur de 'val_accuracy' (paramètre « restore_best_weights » transmis à True). L'ensemble des paramètres disponibles pour le callback "EarlyStopping" est détaillé dans [5].

Le déroulement de l'entraînement est détaillé dans la **figure 5**. On observe qu'après la 11e étape, 3 cycles successifs sans amélioration de 'val_accuracy' ont été détectés, ce qui provoque l'arrêt de l'apprentissage et la sélection des valeurs de paramètres calculées lors du 8e cycle.

Le callback « ModelCheckpoint » constitue une deuxième solution d'amélioration, car il permet une sauvegarde systématique sur disque des paramètres du réseau, après chaque amélioration des performances. Le code est détaillé sur la **figure 6** (voir [6] pour la documentation sur les paramètres).

Un résultat d'exécution est détaillé sur la **figure 7**, avec à chaque cycle le retour d'information sur l'archivage ou non du réseau.

Exemple d'instanciation de la classe générique

Si les callbacks disponibles ne répondent pas au besoin, il est possible d'en créer un « sur-mesure » à partir de la classe abstraite « Callback ». Pour se faire, il faut la dériver en complétant le code des méthodes à utiliser, selon les traitements souhaités (cf. **encadré n°2**). Dans l'exemple ci-dessous, l'objectif est d'arrêter l'apprentissage sur un critère d'atteinte de performance, ici un taux de réussite en validation supérieur à 97,5%. La solution est, à la fin de chaque cycle d'apprentissage, de comparer 'val_accuracy' à 97,5 % et de forcer l'arrêt si la cible est dépassée. Le code à ajouter à la méthode « on_epoch_end » est détaillé sur la **figure 8.**

Grâce aux paramètres « self », « epoch » et « logs » toutes les informations sur le réseau et son fonctionnement sont accessibles à l'intérieur de la méthode :

- « self » fait référence à la classe « Callback » [4], qui comporte 2 attributs : « params » avec les paramètres d'apprentissage

Figure 4

```
# INSTANCIATION de la classe EarlyStopping()
MonEarlyStopping = tf.keras.callbacks.EarlyStopping(
                       monitor = 'val_accuracy',
                       patience = 3,
                       verbose = 1,
                       restore_best_weights=True)

# APPRENTISSAGE du réseau
hist=MonReseau.fit(x=x_train,
                   y=y_train,
                   epochs=15,
                   verbose=2,
                   batch_size=32,
                   validation_data=(x_test,y_test),
                   callbacks=[MonEarlyStopping])
```

Figure 5

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
60000/60000 - 4s - loss: 0.3165 - accuracy: 0.9119 - val_loss: 0.1721 - val_accuracy: 0.9490
Epoch 2/15
60000/60000 - 3s - loss: 0.1555 - accuracy: 0.9552 - val_loss: 0.1295 - val_accuracy: 0.9611
Epoch 3/15
60000/60000 - 3s - loss: 0.1164 - accuracy: 0.9652 - val_loss: 0.1089 - val_accuracy: 0.9666
Epoch 4/15
60000/60000 - 4s - loss: 0.0932 - accuracy: 0.9725 - val_loss: 0.0972 - val_accuracy: 0.9705
Epoch 5/15
60000/60000 - 3s - loss: 0.0782 - accuracy: 0.9759 - val_loss: 0.1006 - val_accuracy: 0.9697
Epoch 6/15
60000/60000 - 3s - loss: 0.0680 - accuracy: 0.9791 - val_loss: 0.0907 - val_accuracy: 0.9729
Epoch 7/15
60000/60000 - 3s - loss: 0.0590 - accuracy: 0.9815 - val_loss: 0.0886 - val_accuracy: 0.9738
Epoch 8/15
60000/60000 - 3s - loss: 0.0524 - accuracy: 0.9837 - val_loss: 0.0841 - val_accuracy: 0.9751
Epoch 9/15
60000/60000 - 3s - loss: 0.0459 - accuracy: 0.9857 - val_loss: 0.0909 - val_accuracy: 0.9737
Epoch 10/15
60000/60000 - 3s - loss: 0.0404 - accuracy: 0.9877 - val_loss: 0.0935 - val_accuracy: 0.9741
Epoch 11/15
Restoring model weights from the end of the best epoch.
60000/60000 - 3s - loss: 0.0377 - accuracy: 0.9882 - val_loss: 0.1014 - val_accuracy: 0.9686
Epoch 00011: early stopping

Évaluation sur la base de tests:
10000/10000 - 0s - loss: 0.0841 - accuracy: 0.9751
Résultats du test: 97.51%
```

Pas de progrès pendant 3 cycles

Meilleure valeur

Figure 6

```
# INSTANCIATION de la classe ModelCheckpoint()
MonModelCheckpoint = tf.keras.callbacks.ModelCheckpoint(
                       filepath = 'MonReseau.keras',
                       monitor = 'val_accuracy',
                       verbose = 1,
                       save_best_only=True,
                       save_weights_only=True)

# APPRENTISSAGE du réseau
hist=MonReseau.fit(x=x_train,
                   y=y_train,
                   epochs=15,
                   verbose=2,
                   batch_size=32,
                   validation_data=(x_test,y_test),
                   callbacks=[MonModelCheckpoint])
MonReseau.load_weights('MonReseau.keras')
```

Figure 7

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
Epoch 00001: val_accuracy improved from -inf to 0.94740, saving model to MonReseau.keras
60000/60000 - 4s - loss: 0.3289 - accuracy: 0.9075 - val_loss: 0.1818 - val_accuracy: 0.9474
Epoch 2/15
Epoch 00002: val_accuracy improved from 0.94740 to 0.96120, saving model to MonReseau.keras
60000/60000 - 3s - loss: 0.1614 - accuracy: 0.9532 - val_loss: 0.1368 - val_accuracy: 0.9612
Epoch 3/15
Epoch 00003: val_accuracy improved from 0.96120 to 0.96410, saving model to MonReseau.keras
60000/60000 - 3s - loss: 0.1213 - accuracy: 0.9645 - val_loss: 0.1183 - val_accuracy: 0.9641
Epoch 4/15
Epoch 00004: val_accuracy improved from 0.96410 to 0.96940, saving model to MonReseau.keras
60000/60000 - 3s - loss: 0.1002 - accuracy: 0.9700 - val_loss: 0.1053 - val_accuracy: 0.9694
Epoch 5/15
Epoch 00005: val_accuracy did not improve from 0.96940
60000/60000 - 3s - loss: 0.0849 - accuracy: 0.9747 - val_loss: 0.1018 - val_accuracy: 0.9684
[...]
Epoch 12/15
Epoch 00012: val_accuracy improved from 0.97220 to 0.97230, saving model to MonReseau.keras
60000/60000 - 3s - loss: 0.0366 - accuracy: 0.9885 - val_loss: 0.0998 - val_accuracy: 0.9723
Epoch 13/15
Epoch 00013: val_accuracy did not improve from 0.97230
60000/60000 - 3s - loss: 0.0339 - accuracy: 0.9894 - val_loss: 0.1087 - val_accuracy: 0.9687
Epoch 14/15
Epoch 00014: val_accuracy did not improve from 0.97230
60000/60000 - 3s - loss: 0.0306 - accuracy: 0.9904 - val_loss: 0.1173 - val_accuracy: 0.9691
Epoch 15/15
Epoch 00015: val_accuracy did not improve from 0.97230
60000/60000 - 4s - loss: 0.0272 - accuracy: 0.9918 - val_loss: 0.1145 - val_accuracy: 0.9705

Évaluation sur la base de tests:
10000/10000 - 0s - loss: 0.0998 - accuracy: 0.9723
Résultats du test: 97.23%
```

Meilleure configuration sauvegardée sur disque, puis rechargée

(nombre de cycles, taille des lots,…) stockés dans un dictionnaire et « model » avec la description du réseau (instance de keras.models.Model). Ici nous utilisons l'attribut « stop_training » qui permet de forcer l'arrêt de l'apprentissage

- « epoch » renseigne sur la valeur du cycle en cours dans l'apprentissage
- « logs » stocke dans un dictionnaire les valeurs courantes des métriques 'accuracy', 'loss', 'val_loss' et 'val_accuracy' ; ici nous exploitons la valeur du taux de réussite en validation pour la comparer à la cible de 97,5%.

L'utilisation dans le programme précédent de la nouvelle classe « MaClasseCallback() » fonctionne en remplaçant la partie apprentissage par le code détaillé sur la **figure 9.**

Les résultats à l'exécution sont détaillés sur la **figure 10.**

Conclusion

Le mécanisme des « callbacks », permet de contrôler de manière extrêmement précise le fonctionnement de l'apprentissage. Il est possible de surveiller son déroulement par l'enregistrement à chaque étape d'informations, comme les valeurs des paramètres ou celles des métriques. Mais il devient surtout accessible d'agir à certaines étapes de l'entraînement pour améliorer les résultats. Nous avons détaillé l'exploitation de la mesure de la performance en validation ('val_accuracy') pour sélectionner le réseau avec les meilleurs résultats. Des traitements similaires peuvent aussi être réalisés en exploitant les autres métriques que sont 'accuracy', 'loss' et 'val_loss'. Une autre piste intéressante à explorer est la modification, avec les callbacks LearningRateScheduler() ou ReduceLROnPlateau(), du taux d'apprentissage utilisé par l'algorithme de descente du gradient. À vos claviers…

Documentation en ligne

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

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

[3] Module "callbacks" de Keras : https://www.tensorflow.org/api_docs/python/tf/keras/callbacks

[4] Code source Python de la classe abstraite "Callback" https://github.com/tensorflow/tensorflow/blob/v2.4.1/tensorflow/python/keras/callbacks.py#L607-L869

[5] Documentation du callback « EarlyStopping » : https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

[6] Documentation du callback « ModelCheckpoint » : https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ModelCheckpoint

[7] Ecole d'ingénierie informatique EPSI (membre de HEP Education) : https://www.epsi.fr/

Pour approfondir

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


```python
# DERIVATION de la classe abstraite
class MaClasseCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs=None):
    if(logs.get('val_accuracy') > 0.975):
      print("\nARRET DE L'APPRENTISSAGE: 97,5% atteint en validation !")
      self.model.stop_training = True
```

Figure 8

```python
# INSTANCIATION de la classe MaClasseCallback()
MonCallback = MaClasseCallback()

# APPRENTISSAGE du réseau
hist=MonReseau.fit(x=x_train,
                   y=y_train,
                   epochs=15,
                   verbose=2,
                   batch_size=32,
                   validation_data=(x_test,y_test),
                   callbacks=[MonCallback])
```

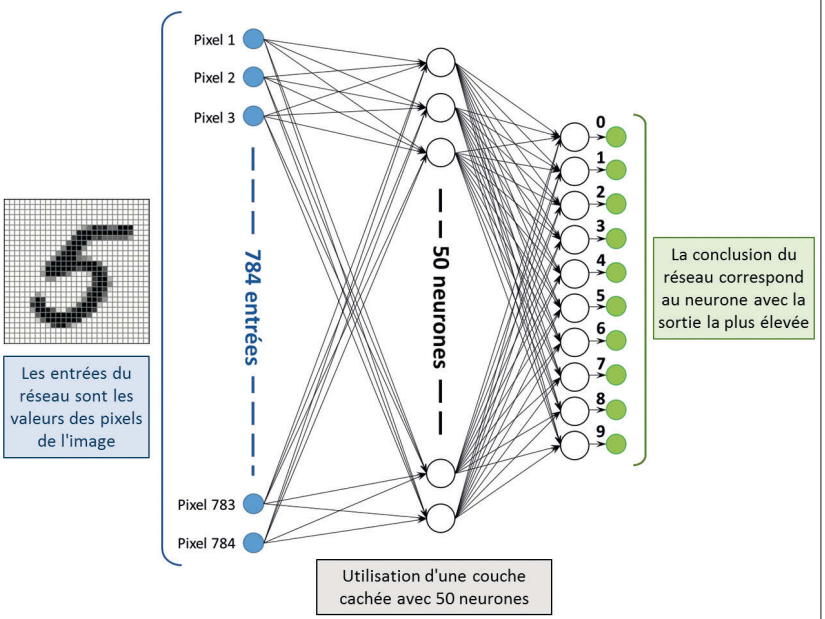
Figure 9

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/15
60000/60000 - 4s - loss: 0.3147 - accuracy: 0.9120 - val_loss: 0.1834 - val_accuracy: 0.9461
[...]
Epoch 8/15
60000/60000 - 3s - loss: 0.0547 - accuracy: 0.9836 - val_loss: 0.0885 - val_accuracy: 0.9733
Epoch 9/15
60000/60000 - 3s - loss: 0.0477 - accuracy: 0.9855 - val_loss: 0.0920 - val_accuracy: 0.9736
Epoch 10/15
60000/60000 - 3s - loss: 0.0425 - accuracy: 0.9869 - val_loss: 0.0849 - val_accuracy: 0.9745
Epoch 11/15
60000/60000 - 4s - loss: 0.0385 - accuracy: 0.9883 - val_loss: 0.0929 - val_accuracy: 0.9731
Epoch 12/15
ARRET DE L'APPRENTISSAGE: 97,5% atteint en validation !
60000/60000 - 3s - loss: 0.0331 - accuracy: 0.9897 - val_loss: 0.0886 - val_accuracy: 0.9756

Évaluation sur la base de tests:
10000/10000 - 0s - loss: 0.0886 - accuracy: 0.9756
Résultats du test: 97.56%
```

Arrêt de l'apprentissage

Figure 10

Encadré n°3: perceptron multicouche utilisé dans le programme



Ce réseau est utilisé pour la reconnaissance de chiffres manuscrits. Une image de 28x28 pixels en niveaux de gris est présentée en entrée du réseau. Celui-ci comporte 10 sorties, chacune associée à la conclusion sur le niveau d'identification d'un des 10 chiffres.

À la découverte de l'algorithme quantique Grover

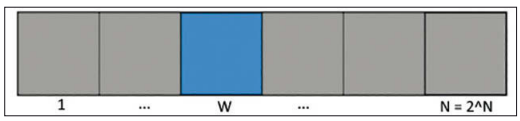
Clément Breuzet

Passionné de science et d'informatique, Clément est actuellement développeur au sein d'Avanade France. Depuis 3 ans, il s'intéresse à la physique quantique et notamment dans son usage dans l'informatique. « J'ai envie d'être un jour un acteur de ce domaine qu'est l'informatique quantique »

Les algorithmes quantiques… une révolution qui va nous permettre d'augmenter drastiquement nos capacités de calcul. Pour démontrer cela, j'ai choisi de vous présenter un des algorithmes phares, Grover. Celui-ci permet grossièrement de rechercher « une aiguille dans une botte de foin » ou plus techniquement ; appliquer une recherche dans une collection non ordonnée. Actuellement, nous effectuons ce genre de recherche de manière linéaire et ce n'est pas très efficace pour un nombre élevé d'éléments. Ici Grover va appliquer une recherche dite quadratique, c'est-à-dire plus le nombre d'éléments est grand, plus l'algorithme sera efficace. Il peut également servir pour obtenir des améliorations du temps d'exécution pour d'autres algorithmes. C'est ce qu'on appelle l'astuce d'amplification d'amplitude.

Recherche non structurée

Supposons qu'on vous donne une liste de N éléments. Parmi ces éléments, il y a un élément avec une propriété unique que nous souhaitons identifier ; nous appellerons celui-ci le gagnant ω. Admettons que tous les éléments de la liste soient gris, à l'exception du gagnant ω, qui est bleu.



Pour trouver la case bleue dans cette collection en utilisant le calcul classique, il faudrait vérifier au minimum N/2 cases, et dans le pire des cas, N cases. Sur un ordinateur quantique, cependant, nous pouvons trouver l'élément marqué en $\sqrt{N}$ itération avec l'astuce d'amplification d'amplitude de Grover. Une accélération quadratique est en effet un gain de temps pour trouver des éléments marqués dans de longues collections. De plus, l'algorithme n'utilise pas la structure interne de la liste, ce qui la rend générique ; c'est pourquoi il fournit immédiatement une accélération quantique quadratique pour de nombreux problèmes classiques.

Création d'un oracle

Dans cet exemple, notre jeu de donnée représente tous les états de base possibles dans lesquels nos qubits peuvent être. Par exemple, si nous avons 3 qubits, nous aurons respectivement 2^3, soit $|000\rangle$, $|001\rangle$, $|011\rangle$, $|111\rangle$, $|100\rangle$, $|101\rangle$, $|110\rangle$, $|010\rangle$.

L'algorithme de Grover résout les oracles qui ajoutent une phase négative aux états de solution. C'est-à-dire pour tout état $|x\rangle$:

$$U_\omega|x\rangle = \begin{cases} |x\rangle & if\ x \neq \omega \\ -|x\rangle & if\ x = \omega \end{cases}$$

Cet oracle sera donc une matrice diagonale où l'entrée correspond à l'élément marqué qui aura une phase négative. Par exemple, si nous utilisons encore nos 3 qubits et que = 101, notre oracle aura la matrice suivante :

$$U_\omega = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Ce qui rend l'algorithme de Grover si puissant, c'est à quel point il est facile de convertir un problème en un oracle de cette forme. Il existe de nombreux problèmes pour lesquels il est difficile de trouver une solution, mais relativement facile de vérifier une solution. Par exemple, nous pouvons facilement vérifier une solution à un sudoku en vérifiant que toutes les règles soient satisfaites.

Pour ces problèmes, nous pouvons créer une fonction f qui prend une solution proposée x et retourne $f_{(x)} = 0$ si x n'est pas une solution ($x \neq \omega$) et $f_{(x)} = 1$ pour une solution valide ($x = \omega$).

Notre oracle peut être décrit de la manière suivante :

$$U_\omega|x\rangle = (-1)^{f(x)}|x\rangle$$

Notre matrice devient donc :

$$U_\omega = \begin{bmatrix} (-1)^{f(0)} & 0 & \cdots & 0 \\ 0 & (-1)^{f(1)} & \cdots & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & (-1)^{f(2^n-1)} \end{bmatrix}$$

Amplification d'amplitude

Maintenant que nous avons une approche mathématique de l'algorithme de Grover, nous allons voir comment celui-ci marche. Avant de regarder une collection d'éléments, nous n'avons aucune idée de l'emplacement de l'élément marqué ω. Par conséquent, toute estimation de son emplacement est aussi bonne que toute autre, ce qui peut être exprimé en termes de superposition uniforme.

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

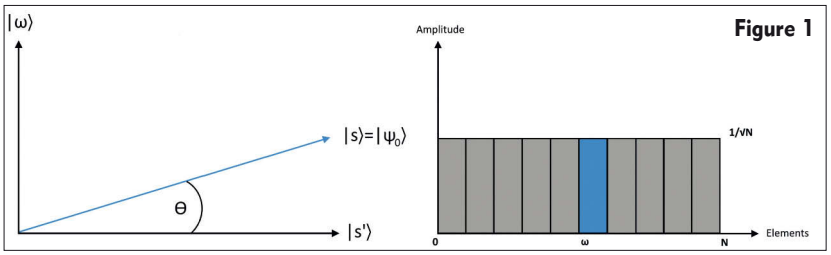


Figure 1

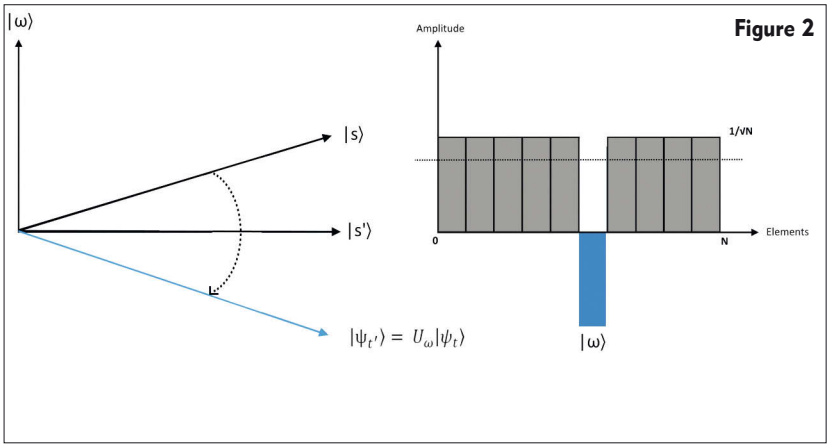


Figure 2

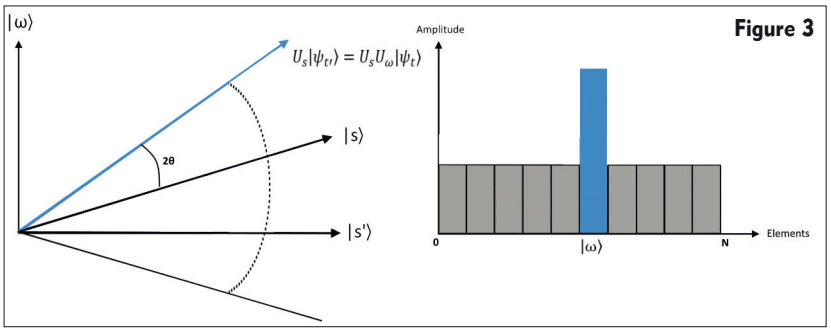


Figure 3

Si à ce stade nous mesurions dans la base standard $\{|x\rangle\}$, cette superposition s'effondrerait, selon la cinquième loi quantique, à n'importe lequel des états de base avec la même probabilité de $\frac{1}{N} = \frac{1}{2^n}$.

Nos chances de deviner la bonne valeur ω sont donc de **1** sur **2n**. Par conséquent, en moyenne, nous aurions besoin d'essayer environ $\frac{N}{2} = 2^{n-1}$ fois pour deviner le bon élément.

La procédure appelée amplification d'amplitude améliore considérablement une probabilité. Nous allons utiliser cette procédure pour amplifier l'amplitude de notre élément marqué ω et réduire l'amplitude des autres éléments. Cela va nous permettre de nous permettre de trouver l'élément marqué avec une quasi-certitude.

On peut également représenter cet algorithme par une représentation géométrique en termes de deux réflexions, qui génèrent une rotation dans un plan à deux dimensions. Les deux seuls états que nous devons considérer sont notre gagnant $|\omega\rangle$ et la superposition uniforme $|s\rangle$. Ces deux vecteurs couvrent un plan à deux dimensions dans l'espace vectoriel C^N. . Ils ne sont pas tout à fait perpendiculaires, car $|\omega\rangle$ apparaît également dans la superposition d'amplitude $N^{-1/2}$. Nous pouvons cependant introduire un état supplémentaire $|s'\rangle$ qui se trouve dans l'étendue de ces deux vecteurs, qui est perpendiculaire à $|\omega\rangle$ et est obtenu à partir de $|s\rangle$ en supprimant $|\omega\rangle$ et en rééchelonnant.

Étape 1 : La procédure d'amplification d'amplitude commence par une superposition uniforme , qui peut être construite de la manière suivante :

$$|s\rangle = H^{\otimes n}|0\rangle^n$$

Figure 1

Le graphique de gauche correspond à un plan bi-dimensionnel parcouru par des vecteurs perpendiculaires $|\omega\rangle$ et $|s'\rangle$ qui nous permet d'exprimer l'état initial par :

$$|s\rangle = \sin\theta|w\rangle + \cos\theta|s'\rangle, \text{ où } \theta = arcsin\langle s|w\rangle = \arcsin\frac{1}{\sqrt{N}}$$

Le graphique de droite représente les amplitudes de l'état $|s\rangle$

Étape 2 : On applique la réflexion de l'oracle U_f à l'état $|s\rangle$

Figure 2

Géométriquement cela correspond à une réflexion de l'état $|s\rangle$ sur $|s'\rangle$. Cette transformation signifie que l'amplitude devant l'état $|\omega\rangle$ devient négative, ce qui veut également dire que l'amplitude moyenne (indiquée par une ligne pointillée) a été diminuée.

Étape 3 : On applique maintenant une réflexion supplémentaire (U_s) sur l'état $|s\rangle$:

$$U_s = 2|s\rangle\langle s| - 1$$

Cette transformation mappe l'état sur $U_s U_f|s\rangle$ et complète la transformation.

Figure 3

Deux réflexions correspondent toujours à une rotation. La transformation $U_s\,U_f$ fait pivoter l'état initial $|s\rangle$ plus près du gagnant $|\omega\rangle$. L'action de la réflexion (U_s) dans le diagramme à barres d'amplitude peut être comprise comme une réflexion sur l'amplitude moyenne. Puisque l'amplitude moyenne a été abaissée par la première réflexion, cette transformation augmente l'amplitude négative de $|\omega\rangle$ à environ trois fois sa valeur d'origine, tandis qu'elle diminue les autres amplitudes. Nous passons ensuite à l'étape 2 pour répéter l'application. Cette procédure sera répétée plusieurs fois pour se concentrer sur le gagnant.

Après t étapes nous serons dans l'état

$|\psi_t\rangle$ où $|\psi_t\rangle = (U_s U_f)^t|s\rangle$

Combien de fois devons-nous appliquer une rotation ? Il s'avère qu'environ $\sqrt{N}$ rotations suffisent. Cela devient logique lorsqu'on regarde les amplitudes de l'état $|\psi\rangle$. On peut voir également que l'amplitude de $|\omega\rangle$ grandit de manière linéaire avec le nombre d'applications. Cependant, puisqu'il s'agit d'amplitudes et non de probabilités, la dimension de l'espace vectoriel entre en racine carrée. C'est donc l'amplitude, et pas seulement la probabilité, qui est amplifiée dans cette procédure.

Exemple simple avec deux Qubits

Voyons d'abord le cas de l'algorithme de Grover pour $N = 4$ qui est réalisé avec 2 qubits. Dans ce cas particulier, une seule rotation sera nécessaire pour faire pivoter l'état initial $|s\rangle$ vers le gagnant $|\omega\rangle$.

Nous savons que $|s\rangle$ ou θ est égal à $\arcsin\frac{1}{\sqrt{N}}$ donc dans notre cas $\theta = \frac{\pi}{6}$

Après t étapes, nous avons

$$(U_s U_\omega)^t |s\rangle = \sin\theta_t |w\rangle + \cos\theta_t |s'\rangle \text{ où } \theta_t = (2t+1)\theta$$

Pour obtenir $|\omega\rangle$ nous devons avoir $\theta_t = \frac{\pi}{2}$ et avec $\theta = \frac{\pi}{6}$ inséré ci-dessus aboutit à $t = 1$.

Cela implique qu'après $t = 1$ rotations notre élément sera trouvé.

Oracle pour $|\omega\rangle = |11\rangle$

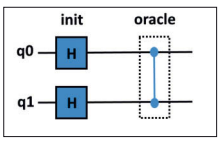
Si on applique la formule précédemment expliquée nous aurons donc :

$$U_\omega|s\rangle = U_\omega \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$$

Ou bien sous forme matricielle :

$$U_\omega = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

On peut reconnaître ici la porte contrôlée Z, ce qui va simplifier grandement notre exemple, car notre oracle sera simplement Z.



Réflexion (U_s)

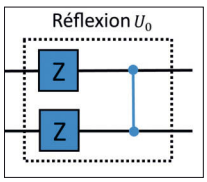
Pour compléter notre circuit, nous devons implémenter une réflexion additionnelle $U_s = 2|s\rangle\langle s| - 1$. Comme il s'agit d'une réflexion sur $|s\rangle$, nous voulons ajouter une phase négative à chaque état orthogonal à $|s\rangle$. Une façon de le faire est d'utiliser l'opération qui transforme l'état $|s\rangle \rightarrow |0\rangle$, que nous connaissons comme étant la porte de Hadamard et sera appliquée à chaque qubit.

$$H^{\otimes n}|s\rangle = |0\rangle$$

Ensuite, nous appliquons un circuit qui ajoute une phase négative aux états orthogonaux à $|0\rangle$.

$$U_0 \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) = \frac{1}{2}(|00\rangle - |01\rangle - |10\rangle - |11\rangle)$$

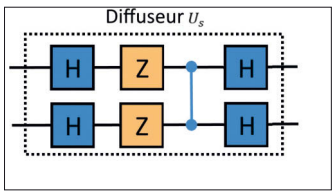
Les signes de chaque état sont inversés à l'exception de $|00\rangle$. Comme on peut facilement le vérifier, une façon de mettre en œuvre U_0 est le circuit suivant :



Et enfin nous appliquons l'opération de transformation à l'état $|0\rangle \rightarrow |s\rangle$. Nous allons encore utiliser la porte de Hadamard H.

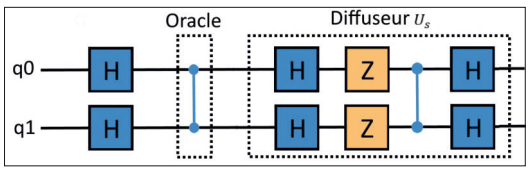
$$H^{\otimes n} U_0 H^{\otimes n} = U_s$$

Le circuit complet pour (U_s) ressemblera donc à ceci :



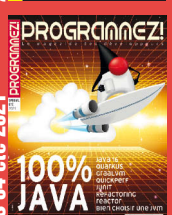
Circuit complet pour $|11\rangle$

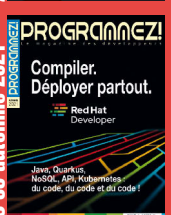
Puisque dans le cas particulier de N=4, une seule rotation est nécessaire, nous pouvons combiner les composants ci-dessus pour construire le circuit complet de l'algorithme de Grover pour le cas $|\omega\rangle = |11\rangle$



Complétez votre collection

voir bon de commande page 43





Jean-Michel Torres
IBM Quantum France
IBM Systems Center | Montpellier

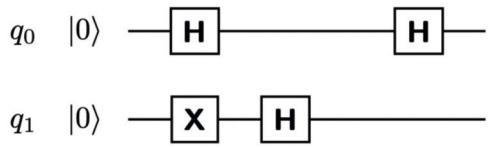
Un algorithme historique du calcul quantique : Bernstein-Vazirani

Créer un algorithme c'est difficile. Comprendre un algorithme quantique n'est pas chose aisée. Les anglo-saxons parlent de « aha moment » quand on vient d'avoir une illumination, une révélation : « Aaah oui ! ». C'est une émotion que je vous propose d'éprouver dans cet article. Comme dans de nombreux cas, le plaisir sera à la hauteur de l'effort, quelques passages vous sembleront rébarbatifs, et pourtant il n'y a rien de difficile. Reprenez les numéros 239 et 240 du magazine Programmez!, votre livre de mathématiques du lycée, un papier et un crayon, c'est parti !

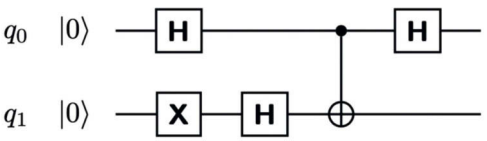
Je vous propose aujourd'hui un autre de ces algorithmes historiques qui sont apparus au commencement de l'ère du calcul quantique, des théoriciens cherchaient à comprendre ce que l'on pouvait espérer faire avec des qubits (quantum bits) et avec les principes du calcul quantique.
Nous sommes en 1992, Ethan Bernstein et Umesh Vazirani, sont parmi les pionniers qui participent au défi lancé au début des années 80 par Richard P. Feynman qui affirmait : « La nature n'est pas classique, mince ! Et si vous voulez simuler la nature, il faudra bien utiliser la mécanique quantique, et je vous assure que c'est un problème merveilleux, car il a l'air loin d'être simple. » Dans leur travail, Ethan Bernstein et Umesh Vazirani utilisent un effet subtil lié à l'intrication de deux qubits : le retour de phase.

RETOUR DE PHASE

Considérons la situation suivante A, on ne s'intéresse qu'à l'état final de q_0 :



De manière évidente (en fait parce que $H^2 = I$) la mesure de q_0 donnera systématiquement la valeur $|0\rangle$
A présent, on rajoute une porte Control-Not ou CNOT (qui produit l'intrication entre q_0 et q_1), pour obtenir la situation B :



Selon la règle « naïve » de la porte CNOT (q_1 change d'état si et seulement si q_0 est à l'état 1) on est tenté d'affirmer que la mesure de q_0, comme dans la situation précédente donnera aussi la valeur $|0\rangle$, puisque cette règle parle de qubit de contrôle et de qubit cible sans laisser entendre qu'il puisse arriver quelque chose au premier. Justement le calcul montre que c'est le cas, voyons cela pour la situation B :
Avant la porte CNOT, le qubit q_0 est dans l'état

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

et le qubit q_1 est d'abord dans l'état

$$X|0\rangle = |1\rangle$$

puis

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Nous allons utiliser la linéarité des opérateurs quantiques (y compris sur plusieurs qubits : $U(x + y) = U(x) + U(y)$) et la multiplication (notée $\otimes$) des états d'un qubit pour former les états de 2 qubits. Cette multiplication a les propriétés habituelles SAUF la commutativité, on peut aussi simplifier l'écriture $|x\rangle \otimes |y\rangle$ en $|xy\rangle$, sans jamais s'autoriser à commuter x et y.
Ainsi l'état des 2 qubits (toujours avant la porte Control-Not) est :

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Que l'on écrit en développant :

$$\frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle)$$

On obtient donc un état composé de quatre termes, lorsque l'on applique la porte CNOT à chacun de ces termes (avec la règle naïve : le second qubit change d'état si le premier est à 1), les deux premiers termes sont inchangés, les deux derniers font changer la valeur du second qubit, on trouve :

$$\frac{1}{2}(|00\rangle - |01\rangle + |11\rangle - |10\rangle)$$

A présent il reste à appliquer la transformation de Hadamard au qubit q_0, chacun des quatre termes va en générer deux nouveaux pour produire la superposition du qubit q_0, la valeur de q_1 ne changeant pas. Cela produira donc 8 termes, il faut faire attention aux signes, en particulier au fait que ,
$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$,

$$CNOT\left(\frac{1}{2}(|00\rangle - |01\rangle + |11\rangle - |10\rangle)\right) = \frac{1}{2}\left(\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) - \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle) + \frac{1}{\sqrt{2}}(|01\rangle - |11\rangle) - \frac{1}{\sqrt{2}}(|00\rangle - |10\rangle)\right)$$

$$= \frac{1}{2\sqrt{2}}(|00\rangle + |10\rangle - |01\rangle - |11\rangle + |01\rangle - |11\rangle - |00\rangle + |10\rangle)$$

$$= \frac{1}{2\sqrt{2}}(2.|10\rangle - 2.|11\rangle) = \frac{1}{\sqrt{2}}(|10\rangle - |11\rangle) =$$

On peut maintenant « factoriser » le qubit q_0 :

$$\frac{1}{\sqrt{2}}(|10\rangle - |11\rangle) = |1\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Et donc en sortie de circuit : le qubit q_0 est à la valeur $|1\rangle$, c'est le retour de phase !

Retenons simplement que dans la situation A q_0 vaut 0 en sortie, mais dans la situation B, avec une porte CNOT qui le lie à q_1 (celui-ci n'étant pas à $|0\rangle$) alors q_0 vaut 1 en sortie.

ALGORITHME DE BERNSTEIN-VAZIRANI

Voici d'abord le problème à résoudre : on interroge un « Oracle » en présentant une donnée x (une suite de longueur n faite de 0 et de 1, par exemple $x = 01001011$), l'Oracle répond 1 ou 0, en fonction d'un secret (qui est également une suite de longueur faite de 0 et de 1). On cherche à connaître en interrogeant l'oracle le moins de fois possible. On sait que la fonction d'évaluation utilisée par l'oracle est :

$$f_s(x) = s_1.x_1 \oplus s_2.x_2 \oplus \dots \oplus s_n.x_n$$

NB : *les « . » sont des multiplications « normales »* $(0.0 = 0, 0.1 = 0, 1.0 = 0, 1.1 = 1)$, *les « $\oplus$ » sont des additions modulo 2* $(0 \oplus 0 = 0, 0 \oplus 1 = 1, 1 \oplus 0 = 1, 1 \oplus 1 = 0)$.

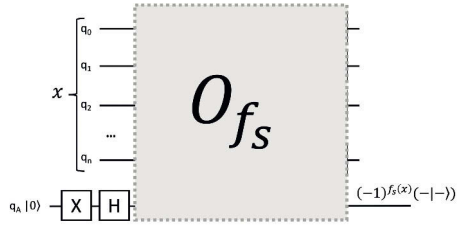
Pour résoudre ce problème de manière classique, il suffit d'interroger l'oracle n fois en utilisant des valeurs de x constituées de n -1 bits à 0 et un bit à 1 placé en première, puis en seconde, puis en troisième… puis en $n^{ième}$ position :

$$f_s(1000\dots0) = s_1$$
$$f_s(0100\dots0) = s_2$$
$$f_s(0010\dots0) = s_3$$
$$\dots$$
$$f_s(0000\dots1) = s_n$$

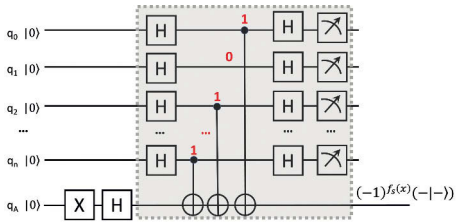
Dans le cadre de l'informatique quantique, avec l'algorithme de Bernstein-Vazirani, il suffit d'une seule interrogation de l'Oracle pour obtenir . Sur le schéma ci-dessous on retrouve l'Oracle correspondant à s : $O_{f_s}, f_s(x)$, , et le qubit auxiliaire q_A préparé dans l'état

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

qui permet d'obtenir le retour de phase vu ci-dessus :



Si on soulève le capot, on se rend compte que l'on peut en effet obtenir s en mesurant les n premiers qubits en sortie, grâce à l'effet de retour de phase décrit plus haut. Notez que toutes les valeurs possibles pour x sont produites par la superposition d'états des n qubits , et vont en quelque sorte se présenter à l'Oracle en même temps. Notez également qu'on n'a pas besoin de mesurer la sortie du dernier qubit tout en bas. La manière d'encoder s consiste tout simplement à relier ou ne pas relier le $k^{ième}$ qubit avec une porte CNOT au qubit du bas selon la valeur de s_k.



CONCLUSION

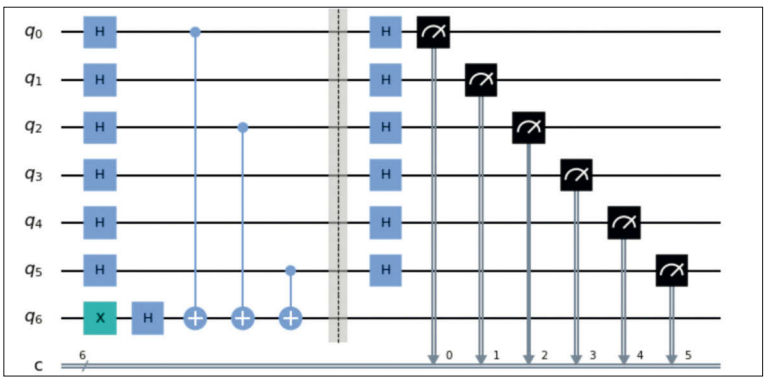
Cet algorithme est-il utile au sens du calcul ? Pas vraiment ! Mais rappelons le contexte : il s'agit de recherche scientifique et ce type de résultat a permis ensuite d'en obtenir d'autres et de construire petit à petit les principes et les outils du calcul quantique.

Les travaux d'Ethan Bersntein et Umesh Vazirani s'inscrivent dans le domaine de la complexité informatique et leur contribution dépasse de loin le cadre de cet article.

Retenons simplement que l'on vient de voir comment résoudre un problème en appelant une seule fois une certaine fonction, là ou un calcul classique aurait nécessité appels à cette fonction.

Cet exemple montre aussi à quel point la notion de calcul quantique nécessite une perspective nouvelle. On perçoit que l'écriture d'algorithme quantiques ne va pas être de l'ordre du « refactoring » !

Vous pouvez essayer cet algo en téléchargeant la bibliothèque qiskit, ou en utilisant IBM Quantum Lab, sur quantum-computing.ibm.com !




```
from qiskit import QuantumCircuit, execute, Aer      # importer qiskit
backend = Aer.get_backend('qasm_simulator')          # choisir la cible d'execution
s = '101001'                                         # le bitstring "secret", par exemple
L = len(s)                                           # sa longueur
qc = QuantumCircuit(L+1,L)                           # qc sera notre circtuit quantique avec L+1 qubit et un
                                                     # registre classige de longueur n pour mesurer s
for i in range(L):
    qc.h(i)                                          # H sur les n qubits

qc.x(L); qc.h(L)                                     # X puis H pour le dernier qubit

for i in range(L):                                   # selon la valeur du bit de s on met une
    if s[i] == '1':                                  # porte CNOT, ou pas
        qc.cx(i,L)

qc.barrier()
for i in range(L):                                   # portes H et mesures
    qc.h(i) ; qc.measure(i,i)

# qc.draw(output='mpl')                              # décommenter pour visualiser le circuit

job = execute(qc,backend)                            # execution et affichage du résultat
res = job.result()
print(res.get_counts())
```

```
{'100101': 1024}
```

RAPPELS

- Un qubit (quantum bit) est représenté par un état général $|\Psi\rangle$ pouvant valoir $|0\rangle$ ou $|1\rangle$ mais également toute combinaison linéaire de $|0\rangle$ et $|1\rangle$ telle que : $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ à condition que $\alpha^2 + \beta^2 = 1$ (ce qui explique les $\frac{1}{\sqrt{2}}$ dans le texte pour les états superposés).
- Ainsi on peut considérer $|\Psi\rangle$ comme un vecteur dans un espace vectoriel dont $\{|0\rangle, |1\rangle\}$ est une base orthonormée.
  De manière immédiate on a $|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}$ et $|1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$.
  Alors si on définit l'opérateur X avec la matrice $\begin{pmatrix}0&1\\1&0\end{pmatrix}$, alors $X|0\rangle = |1\rangle$ et $X|1\rangle = |0\rangle$, cet opérateur correspond au NOT classique pour ces 2 cas.
- L'opérateur de Hadamard, crée la superposition d'états : $\frac{1}{\sqrt{2}}\begin{pmatrix}1&1\\1&-1\end{pmatrix}$,
  alors $H|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ et $H|1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.
- Pour deux qubits : $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ et $|\Phi\rangle = \gamma|0\rangle + \delta|1\rangle$,
  alors $|\Psi\rangle \otimes |\Phi\rangle = (\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle)$
  $= \alpha|0\rangle \otimes \gamma|0\rangle + \alpha|0\rangle \otimes \delta|1\rangle + \beta|1\rangle \otimes \gamma|0\rangle + \beta|1\rangle \otimes \delta|1\rangle$
  $= \alpha\gamma|00\rangle + \alpha\delta|01\rangle$
  $+ \beta\gamma|10\rangle + \beta\delta|00\rangle = |\Psi\Phi\rangle$, l'état du système des deux qubit s'exprime dans un espace vectoriel de dimension 4, dont $\{|00\rangle, |01\rangle, |10\rangle, |00\rangle\}$ est une base orthogonale. On peut ainsi, de cette manière construire les états pour 3, 4 et n qubits.

Benoît Prieur

Développeur indépendant pour sa société Soarthec. Il est par ailleurs l'auteur du livre d'introduction à l'informatique quantique De la physique quantique à la programmation quantique en Q# publié aux éditions ENI en février 2019. Il a également assuré en fin d'année 2019 des cours d'informatique quantique auprès d'élèves ingénieurs de l'ECE Paris.

Introduction à l'informatique quantique avec IBM Qiskit

Le but du présent article est de rappeler succinctement les grands principes de la programmation quantique pour ensuite utiliser une des plates-formes accessibles au grand public permettant d'expérimenter la programmation quantique : IBM QiSkit utilisable complètement en ligne. Qiskit utilise Python, mais peut également utiliser Swift et JavaScript. Ici nous nous contenterons de travailler avec Python en prenant l'exemple des états de Bell correspondant à ce que l'on appelle l'intrication maximale, phénomène dont nous expliquerons le fonctionnement.

Quelques notions de physique quantique

Indifféremment nommé qubit ou qu-bit, le bit quantique représente l'unité de base en informatique quantique à l'instar du bit en informatique. Ce bit quantique est associé à une particule qui se trouve être dans un certain état nommé **état quantique**. Cet état quantique se définit grâce à un phénomène propre à la physique quantique appelé la **superposition quantique**. Avant d'expliquer plus avant ce phénomène, précisons que la superposition quantique s'interrompt quand est effectuée la **mesure quantique**. Ainsi un bit quantique est dans un état quantique qui peut être vu comme une combinaison linéaire des différentes valeurs possibles après mesure. Cette combinaison linéaire des différentes valeurs possibles après mesure concrétise ce qu'est la superposition quantique. Une fois la mesure quantique effectuée, une seule valeur de celle qui était possible précédemment, est conservée.

Histoire de rendre tout cela plus concret, reprenons la célèbre expérience de la pensée, le chat de Schrödinger. Celui-ci va nous permettre d'illustrer chaque notion évoquée puis d'introduire une notation utilisée pour représenter un état quantique, la **notation bra-ket**.

L'expérience du chat de Schrödinger

Erwin Schrödinger énonce les données de son expérience ainsi. Dans une boîte close et opaque se trouvent plusieurs éléments.

- un atome radioactif.
- une fiole de poison mortel.
- un chat.

Si l'atome se désintègre alors la fiole se retrouve ouverte, le poison se diffuse et le chat meurt instantanément. L'atome peut se désintégrer à n'importe quel moment sans qu'à l'extérieur de la boîte on puisse en avoir connaissance. Du point de vue d'un observateur extérieur, il n'y a aucun moyen de savoir si l'atome s'est désintégré et donc de savoir si le chat est mort ou vivant.

D'un point de vue quantique, on peut donc envisager le chat comme étant à la fois *vivant* et à la fois *mort*. On peut en effet établir une combinaison linéaire de l'**état quantique** du chat comme étant une superposition des états finaux après mesure quantique. Ici la **mesure quantique** est ni plus ni moins que l'ouverture de la boîte par un observateur. Une fois ouverte la boîte, le chat est soit mort soit vivant. Les deux états possibles sont en effet *vivant* et *mort*. Tant que la boîte n'est pas ouverte, l'état quantique est donc bien une combinaison des deux **états superposés** *vivant* et *mort*.

En **notation bra-ket**, on écrit un état quantique du chat de la manière suivante :

$$|\psi_{chat}\rangle = a_{vivant}.|vivant\rangle + a_{mort}.|mort\rangle$$

Si on considère les deux états superposés du chat (*vivant* et *mort*) comme étant équiprobables alors l'état quantique du chat s'écrit comme ci-dessous :

$$|\psi_{chat}\rangle = \frac{1}{\sqrt{2}}.|vivant\rangle + \frac{1}{\sqrt{2}}.|mort\rangle$$

Note - *le coefficient multiplicateur devant chaque état dans la précédente équation est égal à la racine carrée de la probabilité associée (ici chaque probabilité est égale à 0.5). En effet :*

$$probabilité(|\text{vivant}\rangle) = probabilité(|\text{mort}\rangle) = (a_i)^2 = \frac{1}{2}$$

Notation bra-ket et mise en évidence de la combinatoire

Prenons un premier exemple avec un seul qubit. Son état quantique peut s'écrire comme une combinaison linéaire des deux états mesurables.

$$|\psi\rangle = \alpha.|0\rangle + \beta.|1\rangle$$

On peut d'ailleurs écrire chacune des deux valeurs mesurables ainsi.

$$|0\rangle = 1.|0\rangle + 0.|1\rangle$$

$$|1\rangle = 0.|0\rangle + 1.|1\rangle$$

Avec deux qubits, on a quatre valeurs possibles mesurables. Ce sont les valeurs suivantes.

$$|00\rangle$$

$$|01\rangle$$

$$|10\rangle$$

$$|11\rangle$$

On a en effet la combinaison linéaire suivante avec les coefficients , , β et α.

$$|\psi\rangle=\alpha.|00\rangle+\beta.|01\rangle+\gamma.|10\rangle+\delta.|11\rangle$$

Comme précédemment on peut exprimer sous forme d'équations chacune des valeurs finalement mesurables.

$$|00\rangle=1.|00\rangle+0.|01\rangle+0.|10\rangle+0.|11\rangle$$
$$|01\rangle=0.|00\rangle+1.|01\rangle+0.|10\rangle+0.|11\rangle$$
$$|10\rangle=0.|00\rangle+0.|01\rangle+1.|10\rangle+0.|11\rangle$$
$$|11\rangle=0.|00\rangle+0.|01\rangle+0.|10\rangle+1.|11\rangle$$

Si l'on passe à trois qubits on aura huit valeurs possibles. De manière générale, une solution à n qubits verra la possibilité de travailler en 2 *exposant* n. On a donc accès à une combinatoire exponentielle qui permet du moins en théorie de résoudre des problèmes très longs à résoudre en informatique classique. Toutefois, à chaque problème, il faut ou faudrait inventer un algorithme quantique *ad hoc*. Ce type d'algorithme prend en entrée un ou plusieurs qubits qui seront ensuite soumis à un circuit quantique. Ce circuit quantique est composé de portes quantiques et se termine systématiquement par une mesure quantique. L'informatique classique et son algèbre de Boole manipulent des portes logiques. En informatique quantique, on utilise des portes quantiques. Les états quantiques étant finalement des états probabilistes, on soumet en général plusieurs fois les mêmes entrées à un circuit quantique. On analyse ensuite la répartition du résultat des 1000 ou 2000 mesures (une mesure par utilisation du circuit quantique).

Avant de commencer à définir et à utiliser des circuits quantiques avec IBM Qiskit, nous allons en quelques lignes expliquer une autre notion de physique quantique particulièrement utile en informatique quantique. L'intrication (ou l'enchevêtrement) quantique.

L'intrication quantique

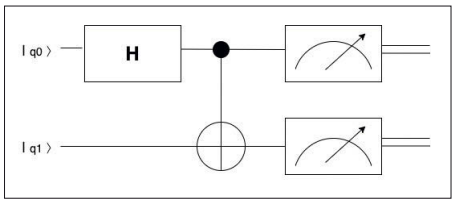
L'intrication quantique est un phénomène physique assez étonnant dont l'existence fut longtemps controversée au sein de la communauté scientifique. Il a été depuis démontré théoriquement et reproduit à plusieurs reprises de façon expérimentale. Il consiste en la mise en évidence d'un lien entre deux particules, pourtant possiblement très éloignées l'une de l'autre. Deux particules A et B qui sont intriquées ont le même état quantique. Si l'une des deux change d'état, l'état de la seconde particule sera le même. Cela implique enfin que le résultat de la mesure respective de deux particules intriquées est le même.

Ce phénomène est fréquemment utilisé en informatique quantique. Pour coder ce qu'on appelle la téléportation quantique ou encore pour simuler les états de Bell qui sera l'objet de notre exemple de code avec Qiskit.

Les états de Bell

Un des moyens d'implémenter l'intrication de deux particules, donc de deux qubits, est de recourir aux états de Bell. Ceux-ci permettent de maximiser l'intrication entre les deux qubits. Ainsi, si on sollicite 1000 fois le circuit quantique qui permet cela, la mesure finale nous indiquera 1000 fois que les deux qubits ont la même valeur.

Un circuit quantique bien déterminé et bien connu permet d'implémenter les états de Bell. Il implique l'utilisation de deux portes quantiques, la porte de Hadamard et la porte C-NOT.



Le schéma précédent prend deux qubits en entrée (q0 et q1) soumis à une porte de Hadamard puis à une porte C-NOT pour ensuite aborder la mesure des deux sorties. En théorie, les deux valeurs mesurées doivent être identiques.

IBM QisKit

Qiskit permet de s'essayer à l'informatique quantique de plusieurs manières. En premier lieu, Qiskit s'installe et s'utilise localement indifféremment sous Linux, Windows ou macOS. L'url suivante détaille la procédure pour installer QiSkit sur votre machine : https://qiskit.org/documentation/install.html

La plate-forme permet également de travailler entièrement en ligne. C'est la solution que nous utiliserons ici. D'une part il est mis à disposition un éditeur graphique de circuits quantiques nommé *Circuit Composer* (nous en dirons un mot) et d'autre part, il est possible de coder et d'exécuter tous ses programmes quantiques en Python dans des *notebooks Jupyter* que nous allons largement utilisée ici.

Une fois défini votre circuit quantique (soit avec *Circuit Composer* soit au sein d'un *notebook Jupyter*) on peut exécuter le programme. Qiskit permet de soumettre le programme à deux types de solveurs quantiques.

- des simulateurs de machines quantiques conçus avec un ordinateur classique qui sont supposés donner un résultat exact d'un point de vue quantique.
- de vraies machines quantiques. L'exécution se retrouve alors dans une file d'attente. En effet, ces machines quantiques sont utilisables par tous les utilisateurs IBM. Cette possibilité est susceptible de donner des résultats partiellement faux, les solveurs quantiques étant particulièrement instables.

Note - *les notebooks Jupyter permettent la programmation interactive faisant alterner du code Python et le résultat graphique ou non de son exécution. L'extension d'un fichier de notebook Jupyter est .ipynb. Un tel fichier peut facilement être partagé, y compris sur des sites de gestion de développement comme Github qui savent afficher un fichier .ipynb.*

Pour utiliser Qiskit en ligne, commencez par vous créer un compte IBM en ligne à l'url suivante.

- https://quantum-computing.ibm.com/

Une fois créé votre compte, vous accédez à votre espace muni d'une barre latérale à gauche de l'écran qui vous permet de naviguer parmi les différents outils mis à disposition. Ci-dessous une copie d'écran de la barre latérale incluant les liens suivants.

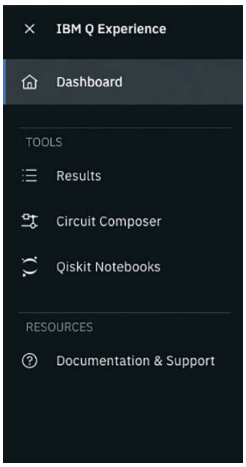


Figure 1

- *Dashboard* qui fait office de page d'accueil.
- *Results* qui permet de consulter les exécutions passées et leurs résultats.
- *Circuit Composer*, qui permet de réaliser un circuit quantique de manière graphique.
- *Qiskit Notebooks*, qui permet de travailler avec des *notebooks Jupyter*. **Figure 1**

Le *Circuit Composer* de Qiskit

Cet outil permet de composer son circuit quantique en faisant glisser les différentes portes quantiques de notre programme. Ainsi en quelques clics on réalise le schéma suivant en faisant glisser une porte de Hadamard (H), une porte C-NOT ainsi que deux portes de mesure. **Figure 2**

Au-dessus du schéma ainsi composé, se trouve un bouton *Run* dans lequel on peut sélectionner un solveur ainsi que le nombre d'utilisations du circuit quantique. Nous choisissons donc de lancer 1024 essais à l'aide d'un des simulateurs quantiques mis à disposition.

Après quelques secondes on obtient un résultat graphique qu'il s'agit d'interpréter. **Figure 3**

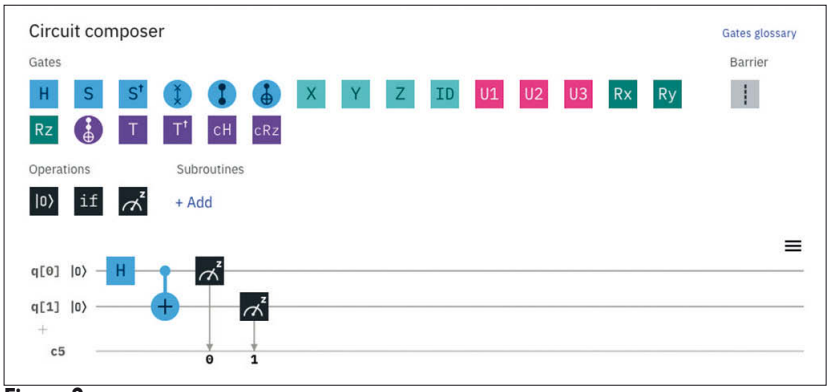


Figure 2

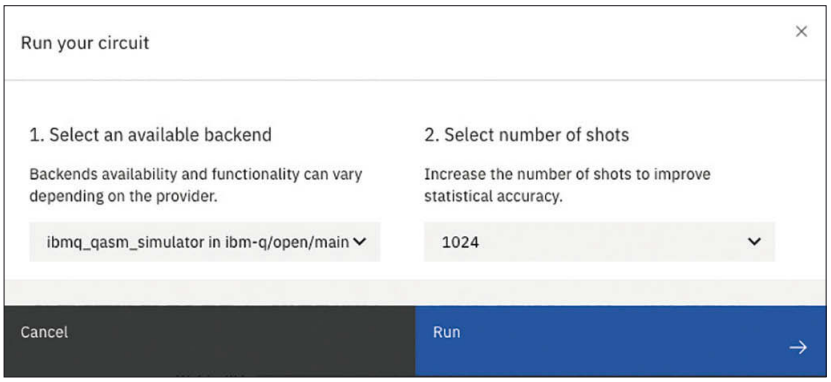


Figure 3

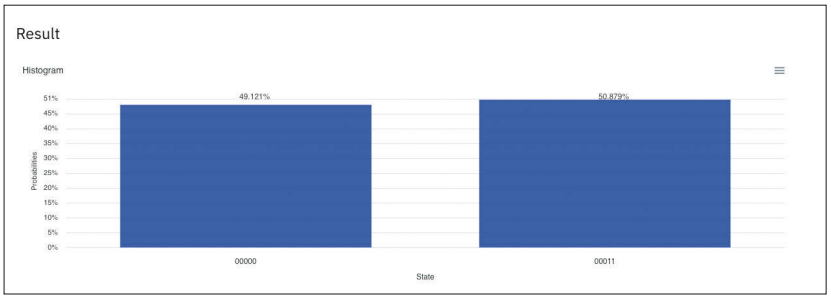


Figure 4

Note – *On n'interprète que les deux derniers digits de chaque résultat. En l'occurrence les trois premiers zéros (000) dans 00011 ne sont pas signifiants ; les deux derniers digits correspondants aux deux mesures de qubits de sortie.*

Figure 4

On voit qu'on obtient 49,121 % des essais (503 essais) qui donnent le résultat suivant dans lequel les deux qubits de sortie sont identiques (à zéro) :

$$|00\rangle$$

On obtient également 50,879 % des essais (521 essais) qui donnent le résultat suivant dans lequel les deux qubits de sortie sont identiques (à un) :

$$|11\rangle$$

C'est dans ce sens que l'on atteint un maximum d'intrication. Les deux mesures sont toujours de valeurs identiques entre elles. En effet, nous n'obtenons jamais les deux situations suivantes :

$$|10\rangle$$

$$|01\rangle$$

Les *notebooks Jupyter* dans Qiskit

Passons à présent à l'utilisation de *Jupyter*. Nous allons reproduire le même exemple, mais cette fois avec du code Python. Nous lancerons une simulation similaire à celle qui précède puis nous exposerons notre programme quantique sur une vraie machine quantique située à Melbourne. Commençons à créer un nouveau notebook Jupyter en cliquant sur le bouton *New Notebook*.

On commence ensuite à coder une première section du *notebook* que nous détaillons tout de suite.

On commence par écrire la clause *%matplotlib inline* nécessaire dès l'instant que l'on travaille avec des *notebooks Jupyter*.

```
%matplotlib inline
```

Puis on déclare les différents modules dont on aura besoin à commencer par qiskit. On importe également numpy et plot_histogram pour avoir un résultat graphique.

```
import qiskit
from qiskit import(
    IBMQ,
    ClassicalRegister,
    QuantumCircuit,
    QuantumRegister,
    QuantumCircuit,
    execute,
    Aer)
import numpy as np
from qiskit.visualization import plot_histogram
```

Puis on définit notre circuit quantique à l'aide de la fonction *QuantumCircuit* dont la documentation est à cette url.
https://qiskit.org/documentation/api/qiskit.circuit.QuantumCircuit.html

On a besoin de deux bits quantiques et en quelque sorte de deux bits classiques correspondant à l'emplacement de la mesure de chaque bits quantiques de sortie.

```
circuit = QuantumCircuit(2, 2)
```

Puis on adjoint une porte de Hadamard sur la première ligne de bit quantique.

```
circuit.h(0)
```

On ajoute ensuite notre porte quantique C-NOT entre la première ligne de bit quantique et la seconde ligne.

```
circuit.cx(0, 1)
```

Puis on ajoute les deux unités de mesure quantique dont on interprétera les résultats.

```
circuit.measure([0,1], [0,1])
```

Pour l'instant on a juste défini un circuit quantique que l'on va dessiner à l'écran avant de l'exécuter.

```
circuit.draw()
print(circuit)
circuit.draw(output='mpl', filename='circuit.png')
```

La copie d'écran de la première section de notre notebook : **Figure 5**

Quand on l'exécute, on obtient deux représentations de notre circuit quantique. **Figure 6**

Nous allons à présent exécuter ce programme quantique. La seconde section du notebook concerne l'exécution du programme sur un simulateur quantique supposé donner un résultat parfait d'un point de vue quantique. La troisième section concerne l'exécution sur une vraie machine quantique.

Voici le code commenté de la deuxième section.

On déclare un simulateur qui sera chargé de l'exécution.

```
simulator = Aer.get_backend('qasm_simulator')
```

On exécute notre circuit quantique pour 1000 essais.

```
job = execute(circuit, simulator, shots=1000)
```

On obtient le résultat de la simulation.

```
result = job.result()
counts = result.get_counts(circuit)
```

Enfin on affiche un histogramme synthétisant les résultats.

```
print("\nRésultats :", counts)
plot_histogram(counts)
```

La copie d'écran de la deuxième section de notre notebook ainsi que son résultat obtenu quasi immédiatement. Les résultats uniquement en 00 ou en 11 confirment que l'on est bien en intrication maximale de manière parfaite (en effet aucun artefact 01 ou 10 n'est présent même de façon minime dans les résultats). Passons à présent à la troisième section qui utilise une vraie machine quantique. On commence par déclarer cette machine quantique localisée à Melbourne.

```
provider = IBMQ.get_provider(group='open')
device = provider.get_backend('ibmq_16_melbourne')
```

Puis on exécute notre circuit quantique sur cette machine.

```
job_exp = execute(circuit, device, shots=1024)
```

Et comme dans la deuxième section on obtient puis on met en forme les résultats.

```
%matplotlib inline
import qiskit
from qiskit import(
  IBMQ,
  ClassicalRegister,
  QuantumCircuit,
  QuantumRegister,
  QuantumCircuit,
  execute,
  Aer)
import numpy as np
from qiskit.visualization import plot_histogram

circuit = QuantumCircuit(2, 2)
circuit.h(0)
circuit.cx(0, 1)
circuit.measure([0,1], [0,1])

# Circuit
circuit.draw()
print(circuit)
circuit.draw(output='mpl', filename='circuit.png')
```

Figure 5

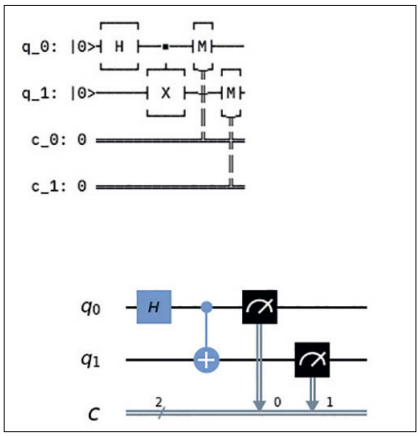


Figure 6

```
Entrée [30
# Machine quantique
provider = IBMQ.get_provider(group='open')
device = provider.get_backend('ibmq_16_melbourne')
job_exp = execute(circuit, device, shots=1024)
result_exp = job_exp.result()
counts_exp = result_exp.get_counts(circuit)
print("\nRésultats expérimentaux:", counts_exp)
plot_histogram(counts_exp)
```

```
Résultats expérimentaux: {'10': 28, '11': 449, '00': 519, '01': 28}
```

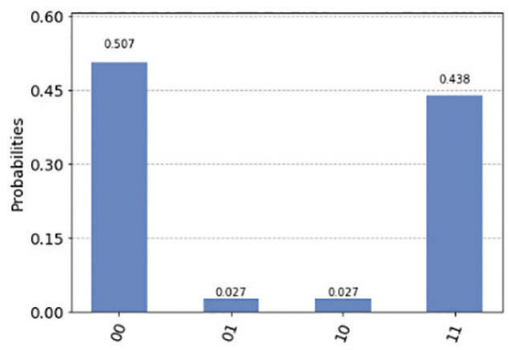


Figure 7

```
result_exp = job_exp.result()
counts_exp = result_exp.get_counts(circuit)
print("\nRésultats expérimentaux :", counts_exp)
plot_histogram(counts_exp)
```

Les résultats ne sont toutefois pas immédiats. Ce nouveau calcul est placé dans la file d'attente de la machine – nous ne sommes pas les seuls à la solliciter. On peut voir peu après son lancement, son état et sa présence dans la file d'attente dans l'onglet *Results*.

Ci-dessus la copie d'écran de la troisième section de notre notebook ainsi que son résultat obtenu après quelques minutes. **Figure 7**

On voit donc que la machine quantique fonctionne plutôt bien, mais a quelques défauts et approximations. Ainsi un peu moins de 6% des tirages ont conduit à obtenir 01 ou 10 ce qui est en contradiction avec le principe même de l'intrication maximale. Le simulateur quantique utilisé précédemment ne produisait pas de tels artefacts. Certes cette exécution montre un défaut (faible), mais elle a été lancée sur une vraie machine quantique ce qui est tout à fait exaltant. À vous de jouer à présent :)

Directives de compilation

PROGRAMMEZ!

Programmez! n°251 - Mars - Avril 2022

Seigneur Sith niveau directeur des rédactions :
François Tonic - ftonic@programmez.com

Contacter la rédaction : redaction@programmez.com

Ont collaboré à ce numéro : **L. Adam (Zdnet), CommitStrip**

Les contributeurs techniques

Loïc Mathieu
Antonella Blasetti
Valériane
Nathalie Seitz
Emilie C.
Romy Alula
Jimmy Kasprzak
Laurent Bossavit
Hervé Boisgontier
Dorra Bartaguiz
Jean-Christophe Riat
Jean-Michel Torres
Benoît Prieur
Clément Breuzet

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) : 55 € pour 1 an, 90 € pour 2 ans. Etudiants : 45 €. Europe et Suisse : 65 € - Algérie, Maroc, Tunisie : 64 € - Canada : 80 € - Tom - Dom : voir www.programmez.com.
Autres pays : consultez les tarifs sur www.programmez.com.

Pour toute question sur l'abonnement :
abonnements@programmez.com

Abonnement PDF
monde entier : 45 € pour 1 an.
Accès aux archives : 20 €.

Nefer-IT
57 rue de Gisors, 95300 Pontoise France
redaction@programmez.com
Tél. : 09 86 73 61 08

Toute reproduction intégrale ou partielle est interdite sans accord des auteurs et du directeur de la publication. © Nefer-IT / Programmez!, mars 2022.

Square

Commencer à développer avec → Square

Développez votre activité en créant des solutions de paiement omnicanales pour nos commerçants.

< En personne >
Terminal API connecte votre application à Square Terminal, le terminal de paiement préféré des commerçants.

< En ligne >
Utilisez le Web Payments SDK pour intégrer des paiements en ligne dans vos applications Web.

< Sur mobile >
Utilisez le In-App Payments SDK pour prendre des paiements depuis vos applications mobiles.

Commencez à
developer.squareup.com/fr

softfluent

Rejoignez **l'ESN** créée par des **développeurs** pour les Développeurs

Vous possédez une ou plusieurs de ces compétences ?

- Azure DevOps
- Cloud
- SQL Server
- Angular
- React
- C#
- CI/CD
- .NET Core
- ASP.NET MVC
- Microservices

Contactez nous !

Retrouvez-nous sur notre page carrière : **softfluent.fr/nous-rejoindre**

Ou contactez Marine, en charge du recrutement chez SoftFluent :

06 69 26 27 87

marine.genetay@softfluent.com

- Conseil
- Expertise
- Partage

softfluent.fr