PROGRAMMEZ!

Le magazine des développeurs

Web 3
.NET 6
ECMAScript
Jamstack
MongoDB

N°252
05/06
2022

M 04319 - 252 - F: 6,99 € - RD

Craftsmanship
Quantique
Carvel
Classes abstraites
Node JS

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

Contenus

Divers

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

PROGRAMMEZ!252

Vers des terres inconnues

Et si on parlait de Web 3 ? Faut-il utiliser le terme Web 3.0, Web3, Web 3 ? Personne n'est vraiment d'accord. Le Web 3.0, tel qu'il avait été évoqué il y a plusieurs années, était une évolution du Web 2.0 avec la multiplication des réseaux sociaux et l'interconnexion toujours plus grandes. Le Web3 serait alors un Internet toujours plus interconnecté et décentralisation. Par décentralisation, on entend que le navigateur ne soit plus au cœur de la navigation de l'utilisateur. Certaines définitions y voient un Internet où nous serions en mesure de mieux contrôler nos données notamment avec la généralisation de la blockchain. Il s'agit aussi d'en finir avec l'importance des fournisseurs technologiques et des récoltes de données. D'autres y voient le Web sémantique ou un Web immersif. Bref, on peut y mettre un peu ce que l'on veut.

Thierry nous plonge dans ce nouveau Web et nous explique ce que nous pourrions y trouver. Une lecture passionnante.

Connaissez-vous vraiment JavaScript et donc ECMAScript qui est le standard officiel du langage ? L'aventure débute en juin 1997 avec le standard ECMA-262. Depuis, les évolutions se succèdent régulièrement. Actuellement, nous sommes à la 10e édition soit ECMAScript 2021. Sylvain, nous prépare toutes les évolutions et les nouveautés jusqu'à aujourd'hui. Un passionnant voyage.

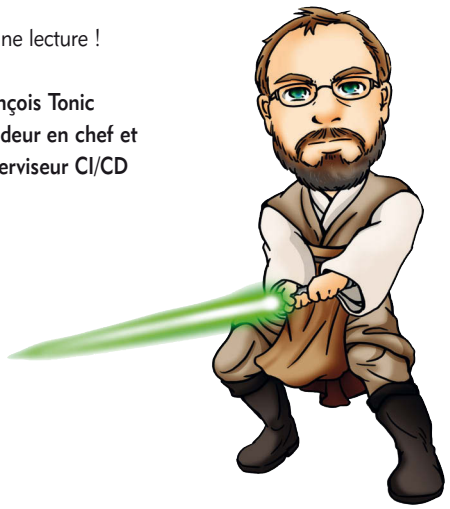
Nous repartons aussi pour le merveilleux monde de .Net 6 et notamment autour du projet MAUI, reboot de Xamarin / Xamarin Forms. Cette fonctionnalité fut retirée à la dernière minute suite à un important retard de développement. Les previews se sont succédées depuis septembre 2021. La GA est toujours attendue pour le 2e trimestre, et par défaut dans .Net 7. Concernant .Net 7, le développement avance. La phase preview a débuté mi-février. Côté Java, OpenJDK 19 a lui aussi débuté son développement. Au moment d'écrire cet édito, seul le portable sur Linux/RISC-V 64 bits était référencé dans les JEP.

Dans ce numéro, nous sommes ravis de partager l'expérience du projet Coding Journey développé par des étudiants d'Epitech. L'idée est « simple » : faire découvrir la programmation de manière ludique et progressive. Franchement, les dévs ont fourni un super travail sur le fonctionnement et l'ergonomie. Bravo !

Nous vous réservons de nombreux autres sujets : craftmanship, les tests automatisés, MongoDB, Carvel, de l'informatique quantique avec Aqasm, un peu de NFT et Scrum Master ! Et nous vous proposons aussi de découvrir un nouveau langage : Opera à ne pas confondre avec le gâteau et le navigateur.

Bonne lecture !

François Tonic
Buildeur en chef et superviseur CI/CD



NOUVEAU !

100 % APPLE
100 % APPLEMANIAC

24 MACHINES QUI ONT FAIT L'HISTOIRE

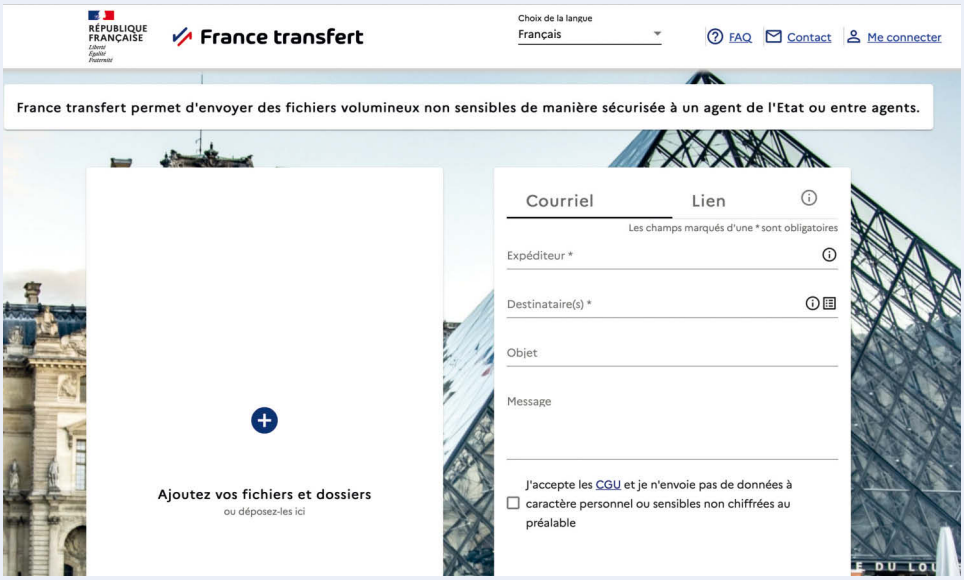
Apple I, Apple II, Spartacus, Cube, Lisa, PowerBook, Macintosh RISC, etc.

150 PAGES

(numéro double + numéro bonus)

*(26,65 € + 6,35 de frais de port)

Commandez sur programmez.com, technosaures.fr, amazon.fr



France Transfert : une alternative française à WeTransfer

Pour transférer des fichiers volumineux, les administrations françaises se heurtaient à une problématique épineuse : difficile de passer par des services comme WeTransfer ou Google Drive et d'expédier sur un cloud américain ou étranger des données parfois potentiellement sensibles. Pour pallier le problème, le gouvernement a dégainé son nouvel outil France Transfert. Destiné en premier lieu aux agents de l'État, le service permet de transférer des fichiers allant jusqu'à 2Go pour un total de 20go. Le tout en ayant l'assurance que les données restent hébergées sur un cloud sécurisé approuvé par l'Anssi. Que demander de plus.

Fuite de données de santé : une amende de 1,5 million d'euros à la clef

La CNIL a épinglé la société Dedalus Biologie pour ses nombreux manquements au RGPD, et a infligé une amende de 1,5 million d'euros à la société. Vous trouvez ça élevé ? La sanction s'explique dans les détails livrés par la Commission : elle explique dans sa délibération que les erreurs de la société ont conduit à la diffusion contenant les données de santé de 500 000 citoyens français au mois de février 2021. Le fichier avait été laissé sur un serveur FTP non sécurisé et librement accessible sur Internet, et ce malgré les alertes de l'Anssi. L'affaire avait fait grand bruit, et la sentence est en conséquence.

Sigfox : la pépite toulousaine finalement reprise par Unabiz

Placée en redressement judiciaire fin janvier, la société Sigfox spécialisée dans les réseaux d'objets connectés a enfin un repreneur. Le tribunal judiciaire de Toulouse a opté pour le projet de reprise présenté par la société Unabiz. Cette société basée à Singapour est elle aussi spécialisée dans le déploiement des technologies IoT (notamment des technologies développées par Sigfox) et fondée par un ancien de Sigfox. Le nouveau propriétaire s'engage à reprendre 110 des 180 salariés de la société et le choix du tribunal s'accorde avec la volonté des représentants du personnel de la société, qui avaient également recommandé de retenir le projet d'Unabiz. Une transition en terrain connu en somme.

Note de la rédaction : *Unabiz a annoncé le versement d'environ 3,3 millions d'€ pour les actifs.*

Raidforums tire sa révérence

Le forum Raidforums s'était taillé une réputation sur le web en devenant au fil des années une des principales plateformes d'échange de données volées. Un exemple ? Le fichier contenant les données de santé de 500 000 citoyens français avait notamment été distribué via des annonces publiées sur ce forum. Actif depuis 2015, celui-ci avait attiré au cours des dernières années l'attention des autorités, à tel point que celui-ci a finalement été mis hors ligne et son administrateur arrêté par la police. Bien évidemment, la relève se prépare déjà.

Starlink perd ses fréquences

Suite à un recours devant le Conseil d'État d'associations écologistes peu enthousiasmées par la 5G et son monde, Starlink se trouve dépourvu de ses fréquences en France. Le Conseil d'État a en effet estimé que l'entreprise américaine d'Internet par satellite avait manqué à ses obligations avant de déployer son réseau, notamment la nécessité de procéder à une consultation du public avant d'ouvrir à cet acteur américain l'accès aux fréquences. L'attribution ayant été cassée, la société d'Elon Musk pourra donc avoir le plaisir de recommencer l'ensemble du processus et redemander à l'Arcep de lui réattribuer ses fréquences… Sauf si Elon préfère décider d'arrêter les frais.

L'Europe veut pouvoir empêcher les transferts anonymes de cryptomonnaie

L'écosystème de la cryptomonnaie est vent debout contre le nouveau projet de régulation européenne voté jeudi 31 mars par le parlement européen. Le projet, connu sous l'acronyme de MiCa, prévoit en effet de durcir les obligations qui pèsent sur les plateformes du secteur et plus particulièrement en les obligeant à fournir des informations concernant les sources et les bénéficiaires des transferts passants par leurs services. Une flopée de nouvelles obligations qui va très probablement tuer l'innovation dans le secteur du point de vue des détracteurs de cette nouvelle loi, qui ont notamment signé une lettre ouverte appelant les instances européennes à alléger leurs projets en la matière.

Compléments de la rédaction

Les NFT ne réussissent pas à tout le monde !

Le NFT du 1er tweet a été acheté pour 3 millions de $ par Sina Estavi. Après un an, l'heureux propriétaire décide d'en séparer pour l'astronomique sommaire de 49 millions de $. Il promet de verser la moitié de la somme à des œuvres de charité. Les premières enchères sont un peu décevantes : à peine 10 000 $! Ainsi va le marché de l'art et des NFT.

A quoi ressemblera Windows 12 ?

Depuis quelques semaines, la question agite une partie des communautés : à quoi pourrait ressembler le prochain Windows ? Dans la continuité de Win11 ? Un OS tout cloud ? Un Chrome OS killer ? Peut être un début de réponse durant la conférence BUILD.

Les partenaires 2022 de

PROGRAMMEZ!

Le magazine des dévs

Niveau maître Jedi

softfluent

Scaleway

The cloud that makes sense

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

Les événements Programmez!

Meetups Programmez!

Les dates du 1er semestre :

10 mai : GitOps sur Kubernetes avec Carvel

28 juin : Ecriture d'un système bancaire simple et distribué en utilisant Axon

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

DevCon #14 100 % GIT

19 mai

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

mai

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
						1
2	3 Le tremplin des speakers / Tours	4	5	6	7	8
9	10 Meetup Programmez!	11	12	13	14	15
16	17	18 SOCRATES / Drôme	19 DevCon #14	20 AFUP Day / Lille	21	22
23	24	25 MixIT / Lyon	26	27	28	29
30	31					

juin

Lun.	Mar.	Mer.	jeu.	Ven.	Sam.	Dim.
		1 Web2Days / Nantes	2	3	4	5
6	7	8	9 AlpesCraft / Grenoble Le camping des speakers Bretagne	10 DevFest Lille	11	12
13 JFTL / Paris	14 France API	15	16	17	18	19
20	21	22	23 Serverless Day / Paris	24	25	26
27	28	29 Hack in Paris / Paris	30 SunnyTech / Montpellier	1 juillet		

A VENIR

- JUG SummerCamp : 9 septembre / La Rochelle
- Cloud Nord : 29 septembre / Lille
- Paris Web : 6 & 7 octobre / Paris
- Volcamp : 13-14 octobre / Clermont Ferrand
- DevFest Nantes : 20-21 octobre / Nantes
- Codeurs en Seine : 17 novembre / Rouen
- SnowCamp : 25-28 janvier 2023 / Grenoble

Merci à Aurélie Vache pour la liste 2022, consultable sur son GitHub : https://github.com/scraly/developers-conferences-agenda/blob/master/README.md

2022 : une bonne année ?

Comme chaque année, les études se multiplient sur les compétences, les carrières et les profils. Nous vous proposons un aperçu sur les derniers rapports de CodinGame, O'reilly et Yerbo. Spoilers : peu de surprises pour 2022, du JS comme on l'aime.

François Tonic

CodinGame a publié son dernier rapport pour les profils recherchés. Nous sommes sur les tendances des années précédentes : fullstack, backend, DevOps, développeur d'application et bien entendu frontend. Pour 2022, les trois principales compétences que les recruteurs cherchent à recruter cette année sont le **développement web**, le **DevOps** et **l'IA / Machine Learning**.

Le développement web reste une valeur sûre du marché pour les recrutements. Les compétences DevOps explosent aussi. Par contre, le développement mobile, l'UI/UX sont en retrait. Étonnement, tout ce qui est IoT, jeux, robototique, programmation fonctionnelle attirent moins, tout comme la sécurité.

Le top des langages recherchés

Pour la quatrième année consécutive, **JavaScript** est le langage de programmation le plus demandé en 2022. Près de la moitié (48%) des recruteurs du secteur tech interrogés recherche des développeurs maîtrisant JavaScript. **Java** et **Python** suivent en deuxième et troisième positions, comme en 2021. Java continue d'être très demandé à cause du patrimoine existant et de la partie mobile. 59% des développeurs interrogés par CodinGame ont déclaré savoir coder en Python et un peu plus de la moitié (55%) maîtrisent Java. TypeScript ne doit pas étonner. Il arrive cinquième et il est le langage par défaut d'Angular.

Si les développeurs déclarent connaître C++, PHP, C, les entreprises embauchent beaucoup moins de compétences sur ces langages. Kotlin reste très en retrait dans l'étude.

Sur les frameworks, nous ne voyons pas de réelles surprises pour les prochains mois et sur les compétences recherchées. La popularité de JS explique la demande pour les frameworks JS. Nous retrouvons les classiques : React, Node, Angular, Vue. .Net représente un quart des frameworks recherchés. Spring ne doit pas surprendre par le poids de Java.

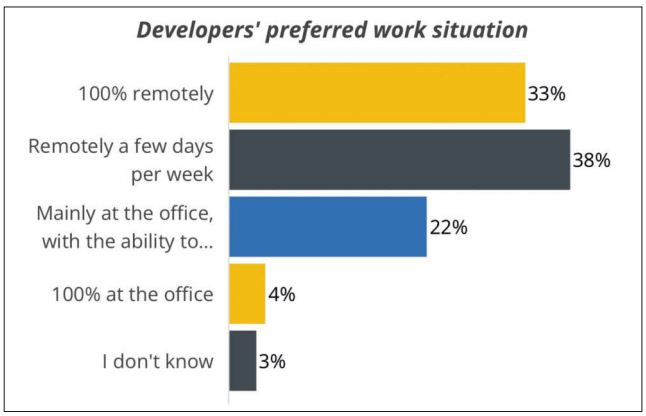
Un constat semble s'imposer : on manque de développeurs. Les formations se multiplient, notamment en filière courte. De plus en plus d'entreprises ne misent plus sur le CV ou le diplôme, mais plus sur les compétences réelles des candidats. Ce discours est sympathique, mais il a des limites. Il ne faut pas croire que l'entreprise recrute à l'aveugle.

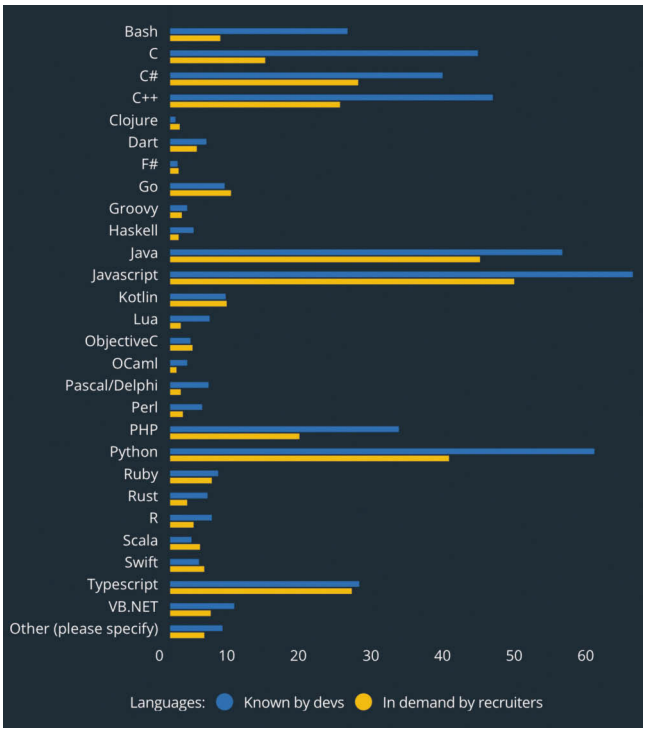
Nous avons discuté avec plusieurs ESN, pure player ou non. La réalité est parfois sévère : 8 candidats sur 10 sont rejetés. Les raisons sont multiples : profil non adapté, compétences annoncées mal maîtrisées, salaire demandé trop élevé. Oui certains profils sont difficiles à trouver, mais la difficulté de recrutement n'est pas uniforme. Si le télétravail s'est brutalement imposé, il ne faut pas pour autant exiger du 100 % télétravail. Dans nos métiers ce n'est pas possible. Rien ne remplace la rencontre avec les équipes, pour s'intégrer réellement à l'entreprise ou encore quand vous êtes en mission. À un moment donné, il faut voir le client…

Quelles tendances pour l'année ?

Nous ne voyons pas de grands changements pour 2022. JavaScript, Angular, Java, C#, mobile resteront des valeurs sûres. PHP est un peu en perte de vitesse, sauf sur les solutions de type CMS. Python confirmera sa bonne forme. Côté salaire, nous ne voyons pas beaucoup de changements.

Les freelances peuvent profiter de la demande soutenue de développement. On verra une fois de plus les métropoles et régions les plus attractives attirer les profils : Ile de France, Auvergne – Rhône-Alpes, Occitanie.





Yerbo : une étude montre-t-elle l'épuisement des équipes ?

Épuisements, tensions, charges de travail, télétravail, turn-over, burn-out, etc. Si on lit l'étude de Yerbo, il y a de quoi être pessimiste et changer immédiatement de métier. Cette étude porte sur 36 200 salariés du monde la tech répartis dans 33 pays.

Selon cette étude, 42 % des salariés veulent / envisagent de changer de société dans les six mois. C'est le pourcentage clé à retenir. Pour la plateforme Yerbo, il s'agit de la preuve d'un accroissement du burn out. Comment serait-on arrivé à cet extrême ? Les explications avancées sont multiples : multiplication des heures de travail, stress lié au projet, tensions dans les équipes, délais de plus en plus courts, effets néfastes du télétravail, équilibre vie pro et vie perso difficile à trouver.

L'explosion des demandes en services numériques liés au Covid a nécessité une nouvelle charge de travail et une adaptation à marche forcée des équipes à cause des confinements. Le télétravail imposé a un double effet que l'on oublie beaucoup trop souvent : le télétravail est une bonne chose pour limiter le temps de transport et le stress des transports, mais l'effet inverse est la difficulté pour des personnes à s'organiser et à télétravailler avec rigueur. Et c'est l'un des travers du télétravail : se laisser submerger. Un cadre strict est nécessaire pour éviter les dérives. Mais surtout, nous sommes isolés. L'Homme, par définition, est sociable et a besoin de son environnement. Et le télétravail à 100 % n'est pas une bonne chose pour les personnes et les entreprises.

L'étude met en avant la confusion vie privée et vie professionnelle et que nous ne prenons pas assez de temps pour nous. Cet argument n'est pas nouveau. Cela fait 20 ans que nous entendons la même chose : comment couper de son activité pro. Insidieusement, on répond à des mails, on termine un petit développement, et on se retrouve à travailler le soir, et le week-end. Le télétravail n'a fait qu'accentuer le phénomène.

Du stress, des tensions ? L'accélération du numérique dans les entreprises, la multiplication des projets, la tension dans les équipes à cause des problèmes de recrutements, pèsent forcément sur les développeurs et les équipes techniques. C'est un réel problème : des délais toujours plus courts, des spécifications nébuleuses, manque de cohérence des équipes, un tech lead ou un CTO dépassé ou ayant du mal à structurer son action, etc. Quand une entreprise n'a pas les effectifs nécessaires, le risque est compensé avec les équipes existantes. Mais cela démontre aussi des problèmes d'organisation.

L'autre stress vient aussi de la technologie elle-même. Un développeur peut être démotivé par des projets peu intéressants, des projets routines, sans technologies innovantes. Ce désintérêt peut aussi venir par le manque de cohésion des équipes, le manque de stimulation entre développeurs, le manque de participation à des meetups, conférences, à des projets open source ou encore pour écrire des contributions à Programmez!. Sans oublier, la culture d'entreprise et comment on valorise les équipes.

Une des difficultés pour les équipes RH et les entreprises est que la motivation varie selon chaque personne. Basiquement, vous avez deux grands profils :

- le développeur passionné qui aime la techno et le code,
- le développeur qui voit le job plutôt comme un job alimentaire : il fait le travail uniquement pour le salaire, le reste ne l'intéresse pas.

Le turn-over est une question récurrente notamment en ESN : la concurrence est forte pour trouver les bons profils. Le turn-over a toujours existé et existera toujours. On estime que la rotation des développeurs et des équipes techniques varie entre 20 et 30 %, ce qui est énorme. Plus le pourcentage est élevé, plus il y a un problème structurel dans l'entreprise. Ce turn-over ne concerne pas tous les profils.

AUX ÉTATS-UNIS : LES SALAIRES S'ALIGNENT SUR LES LANGAGES « RARES »

La comparaison avec les États-Unis est toujours délicate à cause d'une fiscalité très différente. O'Reilly a publié en septembre dernier son 2021 Data / AI Salary Survey. Ce qui peut étonner est une valorisation forte de langages que l'on voit assez peu en France :
+ 170 000 $ / an : Scala, Go, Rust
+ 150 000 $: Objective-C, Erlang, Julia, Swift, F#, Clojure, Lua, Groovy, Ruby, Java

Par contre des langages tels que PHP, C#, Java se situent entre 135 et 149 000 $ / an, ce qui est déjà un excellent salaire. Une partie de cette dynamique peut s'expliquer par la dynamique autour de l'IA, du machine learning.

Pour CodinGame, le salaire moyen en France pour un développeur sera de 47 600 $. Les États-Unis, la Suisse et le Canada sont en tête. La Belgique est devant la France au salaire moyen.

Quelques annonces de recrutements

Groupe SII : 1 950 postes dont 1/3 Paris et 2/3 en province. Profils très variés (.Net, PHP, DevOps, Big Data, etc.)

Sopra Steria : 3 800 postes dont 80 % jeunes diplômés. 20 % sur Paris, 80 % en province

Blue Soft : 130 postes

Coding Journey

Dans le cadre d'un projet de fin d'étude chez Epitech Technology – école d'expertise informatique et d'innovation – nous avons monté une équipe de 9 étudiants développeurs afin de créer un jeu vidéo qui permet de s'initier à la programmation dès le collège : Coding Journey. Après plus de deux années de réflexions et de développement, le jeu est aujourd'hui accessible en ligne gratuitement pour toute personne souhaitant entrer dans l'univers du code.

Figure 1

Coding Journey, c'est quoi ?

Coding Journey est un jeu PC (disponible sous Windows, macOS et Linux) qui permet d'apprendre à coder dans différents langages de programmation (C, PHP et Python sont disponibles actuellement) au travers de différentes aventures. Chaque aventure se focalise autour d'un thème (un langage ou un ensemble de notions spécifiques) et se divise en plusieurs niveaux dans lesquels le joueur apprend à contrôler son personnage via du code.

Concrètement, un niveau a pour but d'enseigner une notion de code, en commençant par l'expliquer au joueur puis en lui demandant de la coder dans un éditeur dédié. Grâce à différentes aides et documentations, le joueur doit coder les actions de son personnage pour atteindre des objectifs. Par exemple, dans les premiers niveaux du jeu, le joueur découvre comment déplacer son personnage à une position donnée en utilisant une fonction et des coordonnées X et Y.

Par la suite, il va découvrir progressivement les variables, les boucles, les conditions et bien d'autres notions.

Via l'interface principale du jeu, il aura même la possibilité de personnaliser son environnement, en achetant des éléments, et équipant ses personnages, en installant des langages de programmation, en changeant d'univers de jeu…

En jouant à un jeu d'aventure dans lequel il réalise des quêtes (affronter des ennemis, collecter des objets…), le joueur assimilera des notions de code tout en s'amusant, c'est là tout l'intérêt de Coding Journey. **Figure 1**

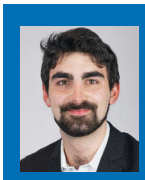
Quelles ont été les motivations du projet ?

En tant que développeurs, nous nous rendons compte aujourd'hui que la programmation est une compétence nécessaire dans tous les domaines. D'après notre propre expérience, nous avons pu constater que les jeunes n'apprennent pas à coder assez tôt, et l'apprentissage est souvent trop scolaire. Notre idée, qui a été pensée entre septembre et novembre 2019, a donc été de diffuser nos compétences par une solution ludique accessible à tous, y compris aux plus jeunes. L'idée est de faire comprendre la logique et les notions fondamentales pour que chacun puisse avoir les bases en programmation, sans avoir l'impression de travailler pour les acquérir.

Certes, des solutions similaires existent (Scratch, CodinGame, Human Resource Machine, ou plus récemment Swift Playground et Twilio Quest), mais notre ambition a été de créer une solution qui soit avant tout un jeu et dans lequel les joueurs ont la possibilité d'apprendre des langages utilisés partout en entreprise.

Comment le projet a-t-il été développé ?

Dans un premier temps, nous avons développé un prototype en décembre 2019, pour montrer la faisabilité du projet, et pour pouvoir le faire tester à quelques personnes. Nous avons alors



Jérémy BRUYERE

jeremy.bruyere@epitech.eu

Étudiant à Epitech Technology (Lyon) et désormais employé chez BASSETTI à Grenoble sur de l'Intelligence Artificielle en R&D, j'ai travaillé sur Coding Journey en tant que chef de projet. En plus de tâches techniques telles que la gestion de l'intégration continue et l'intégration des langages de programmation dans le jeu, je me suis occupé d'organiser le projet, l'équipe et la communication autour du projet pour assurer une progression régulière qui s'accorde avec le besoin des utilisateurs.

Léo LHUILE

leo.lhuile@epitech.eu

Étudiant en dernière année à Epitech Technology (Lyon), je suis employé en tant que développeur fullstack chez Taqt depuis maintenant 1 an. Durant la conception de Coding Journey, j'ai été le responsable du développement sous Unity, aussi bien sur le jeu que sur l'éditeur de niveau. Je me suis principalement occupé de la mise en place de l'architecture du projet, notamment avec la gestion de sous-modules, la conception d'outils pouvant simplifier le développement de tâches futures ou encore le développement de nouvelles fonctionnalités relatives au jeu.

développé un mini-jeu 2D en C++ avec les frameworks Qt et SFML. Le joueur pouvait alors programmer les actions de son personnage en pseudo-code ou avec des « workflows » d'actions dans 5 niveaux.
Nous avons pu le montrer, et même remporter 2 concours de pitch grâce à ce prototype. En tant que vainqueurs, nous avons été soutenus pour repenser l'architecture du projet, son gameplay et son identité graphique.
Le développement du jeu en lui-même s'est déroulé entre mars 2020 et mai 2021. Nous avons réparti notre équipe au sein des différents modules à développer : le jeu (désormais en 3D, en C# et Unity), le site internet (dont la dernière version est en VueJS et en Python Flask), le serveur (d'abord en PHP Symfony puis en C++) et enfin l'éditeur de niveaux (logiciel interne également en C# et Unity qui nous permet de construire le contenu du jeu). Pour être le plus efficace possible, nous avons travaillé sur des sprints de 3 à 4 mois, en intégrant et en déployant les différentes parties au fur et à mesure via de l'intégration continue avec l'outil GitLab CI.
[vidéo Beta] :
https://www.youtube.com/watch?v=FXI-M2g-V0U

Comment le projet a-t-il été testé et amélioré ?

Afin de pouvoir tester le projet significativement, nous avons assuré une communication autour de Coding Journey sur les réseaux sociaux, par email, sur notre site internet, et par le biais d'Epitech qui nous a notamment permis de présenter le jeu sur des salons et des journées portes ouvertes. Nous avons ainsi publié une version bêta en juin 2021 et l'avons fait tester à une trentaine d'utilisateurs de tout profil. Certains étaient des jeunes débutants (notre cible principale), et d'autres des développeurs plus expérimentés qui ont pu nous conseiller sur la manière dont le code est enseigné au sein du jeu.
Le jeu a été bien accueilli par les bêta-testeurs, plus de 75% d'entre eux ont trouvé cette solution pertinente pour apprendre la programmation. En revanche, cette version a été critiquée pour des manquements en termes d'explications et en termes d'expérience utilisateur. Nous avons passé les mois suivants à améliorer le jeu selon les demandes utilisateurs, afin de sortir une version 1.0 en novembre 2021.

Quel a été le résultat de la première version du jeu ?

La première version a pu être publiée en novembre 2021. Elle est disponible en ligne gratuitement sur notre site internet, nous prenons toujours en compte les retours depuis lors.
Cette version a été très bien reçue auprès d'Epitech Technology. En effet, nous avons remporté Epitech Expérience – concours final de tous les projets de fin d'étude Epitech Technology – au niveau local sur la ville de Lyon. Nous faisions alors partie des 15 finalistes (sur un total de 109 projets) à pouvoir participer au niveau national. Coding Journey est le seul projet de la catégorie « Games » à avoir accédé à la finale d'Epitech Expérience 2022.
[vidéo Epitech Expérience] :
https://www.youtube.com/watch?v=_pW22Lsbgas

Qu'est-ce que le projet a apporté ? Et qu'est-ce qui est envisagé pour la suite ?

Si nous espérons que Coding Journey apporte beaucoup aux joueurs en termes de connaissance, il nous a apporté encore plus à nous, étudiants meneurs du projet, sur de nombreux aspects : nous avons pu progresser techniquement en apprenant de nouvelles technologies, mais nous avons progressé aussi grandement sur des aspects fonctionnels tels que la gestion globale d'un projet (l'idéation, la prise de besoin, la faisabilité, la communication, le financement…) et aussi la gestion d'équipe (en sachant que nous avons passé une année de développement en étant répartis dans 6 pays différents pour notre année d'étude à l'étranger). Ce sont des compétences qui nous seront à tous très utiles dans nos métiers futurs.
Maintenant que nous finissons nos études, une partie de l'équipe va se concentrer sur sa professionnalisation en entreprise. Cependant, nous sommes tout de même quelques-uns à continuer d'avancer sur le projet.
Le point le plus important pour la suite réside en la refonte de plusieurs aspects techniques du jeu. En effet, nos connaissances sur Unity ont évolué et nous voyons maintenant que des aspects sont à améliorer par souci de performance. Nous pensons principalement à la gestion des données des différents personnages, des univers ou encore des niveaux en eux-mêmes. À cela viennent s'ajouter quelques améliorations côté UI et UX pour une utilisation plus simplifiée qui fournissent tout de même les outils nécessaires pour progresser.
Nous prévoyons également des ajouts de contenu tels que de nouveaux langages ainsi qu'un langage visuel par "bloc" (similaire à Scratch), un mode multijoueur, ou encore la possibilité de jouer plusieurs personnages en même temps. Nous envisageons aussi une version mobile pour les joueurs qui souhaitent jouer régulièrement, afin d'apprendre en étant dans les transports par exemple. De nouveaux niveaux vont voir le jour afin d'apporter des approfondissements ou de nouvelles notions. Nous penchons aussi sur un nouveau "mode de jeu" qui ne nécessiterait pas d'écrire du code, mais se présenterait sous la forme d'un quiz où chaque réponse déterminerait l'action du héros dans son aventure.
Concernant le site internet, il va également être repensé, avec notamment l'ajout d'une documentation et d'un forum qui permettra aux joueurs de s'entraider en se donnant des astuces de code par exemple.
Les mises à jour seront publiées et partagées sur nos réseaux pour continuer de recenser des avis. Enfin, nous voudrions aussi concrétiser quelques partenariats afin de pouvoir développer d'autres aspects du projet final, comme le suivi des joueurs par des professeurs. Une fois que cette dernière phase sera achevée, nous pourrions officiellement publier le jeu sur des plateformes telles que Steam.

Contacts
Email : eip.coding.journey@gmail.com
Site internet : www.codingjourney.fr
Facebook : www.facebook.com/journeycoding

1, 2, 3, ce sera toi le Scrum Master !

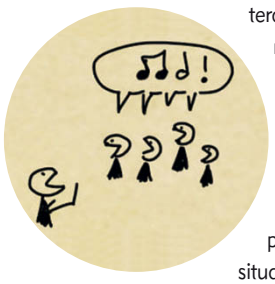
Que nous soyons adeptes de l'agilité sous chacune de ses formes, ou bien complètement newbee, nous avons pu faire face à des pratiques bien étranges lors de la mise en place d'équipes dites « agiles ». Parce que les frameworks connus, ou bien les adaptations que nous en faisons ne définissent pas une méthode à suivre à la lettre, mais plutôt une proposition de bonnes pratiques, certaines organisations d'équipes se cherchent, tâtonnent et parfois, se perdent.

Fanny KLAUK
Accompagnatrice agile chez Apside TOP
Co-créatrice de TADx
@klf37

L'identité d'équipe : pour quoi ?

Un exercice que j'affectionne quand je débute mon observation dans une mission d'accompagnement agile, que ce soit en tant que formatrice, scrum master ou facilitatrice, c'est de me faire une idée de l'identité de l'équipe. On se rend très vite compte de son existence ou non, en fonction de la cohésion observée, de l'entraide et de la légitimité qu'elle se donne aux yeux des autres équipes : forme-t-elle une unité bien précise et visible ? a-t-elle des habitudes qui lui sont propres - un café Teams tous les lundis matin ou des sucrédises régulières (voir la définition latine de dulciolum, NDLR) ? Même nouvellement formée, une équipe en est une lorsqu'elle partage quelque chose de commun malgré les spécialités de chacun et les désaccords constructifs que l'on peut rencontrer.

J'ai eu la chance, dans une expérience récente, de participer à la création d'une nouvelle équipe, et c'est donc par là que j'ai commencé notre aventure commune : nous créer une identité. On se donne un nom (qu'on assumera !), on pousse parfois jusqu'à se choisir une mascotte ou une identité visuelle, on explique nos préférences, nos différences, le "qui aime faire quoi ?", mais surtout, on se définit des valeurs communes : la qualité passera par les tests, on évitera d'utiliser tel outil pour nous faciliter la vie, on écoutera cette chanson à chaque bug en recette… et untel sera notre facilitateur.



Un facilitateur ? pas besoin !

Le besoin d'un facilitateur n'est pas toujours évident, mais il y a des situations qui parlent d'elles-mêmes : s'il s'agit d'une nouvelle équipe qui se rencontre pour la première fois et qui n'a jamais travaillé ensemble ; si l'équipe est composée de nouveaux venus dans l'informatique, sans connaissance de l'agilité, sans référence à un TechLead possible pour les orienter dans les bonnes pratiques de développement ; si l'équipe en place depuis longtemps et qui fonctionnait très bien jusque-là, fait face à des problématiques jamais rencontrées… La question d'avoir un facilitateur ou non prend tout son sens.

Mais même dans ces cas, j'ai vu le refus catégorique. « Il n'y a pas besoin de scrum master, un TechLead et une équipe ça suffit » dixit le coach. Alors d'accord, pourquoi pas, à condition que la situation dans l'équipe et tout autour de l'équipe le permette.

Ce sera toi TechLead !

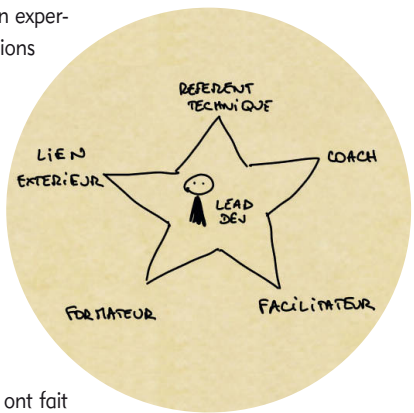
Certains retours d'expériences m'ont montré que ça marche : d'abord le mien. Dans les équipes qui se connaissent et qui fonctionnent bien ensemble depuis un certain temps, le TechLead joue ce rôle de facilitateur à merveille : il est référent dans les bonnes pratiques de développement, de tests, il est souvent le peer-reviewer privilégié, il passe un certain temps à prendre de l'information en dehors de l'équipe… Il a donc un rôle qui le prédestine à être le liant de l'équipe au besoin.

Ensuite, autour de moi, j'ai eu la chance de rencontrer et d'assister aux conférences de Baptiste Lecocq - Scrum est une fenêtre (ou bien un mur) - à Tours JUG 2019 et de Lise Quesnel – Lead Dev, 3 ans d'xp, et alors ? – à Jug SummerCamp 2021.

J'ai littéralement adoré la façon de faire de Baptiste dans son équipe en tant que TechLead : en mode routine, le rôle de facilitateur n'avait pas de grande utilité, et il était donc « absorbé » par chacun des membres de l'équipe, habitués aux process et interactions hors équipe de réalisation. Cependant, il se ré-attitrait ce rôle lors d'occasions bien précises comme l'arrivée d'un nouveau développeur par exemple. Tiens tiens… pour reconstruire l'identité d'équipe, ses valeurs, son "qui fait quoi et comment ?", avant de revenir à la « routine ».

Un des points que je me suis noté de la conférence de Lise, c'est que sous l'étiquette de TechLead pouvait se cacher bien des rôles dont… celui de facilitateur, de coach et de formateur. Mais ce qui me reste en tête, c'est que tout le temps passé avec ces casquettes, à gérer les problématiques de l'équipe, à mettre en relation les bonnes personnes, à (re-) définir les bonnes pratiques et les axes d'amélioration, c'est autant de temps non passé sur le maintien de son expertise technique et de sa prise de recul sur les décisions techniques à prendre.

De ces retours, j'en tire un point de vigilance vital pour l'avenir des équipes sous ce mode de fonctionnement : la gestion du temps et des priorités en est affectée. Il me semble qu'il est donc nécessaire d'en avoir une certaine expérience pour ne pas se noyer ni devoir faire un choix cornélien entre deux casquettes différentes.

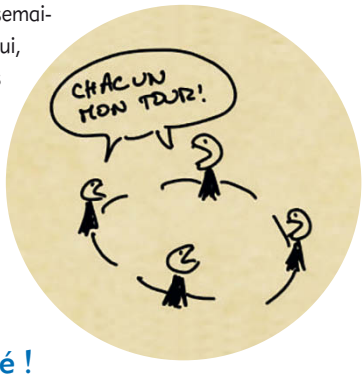


Un rôle tournant ?

J'ai rencontré plusieurs situations différentes qui ont fait naître le rôle tournant du facilitateur au sein d'une équipe. La première situation était que, faute de scrum master dédié,

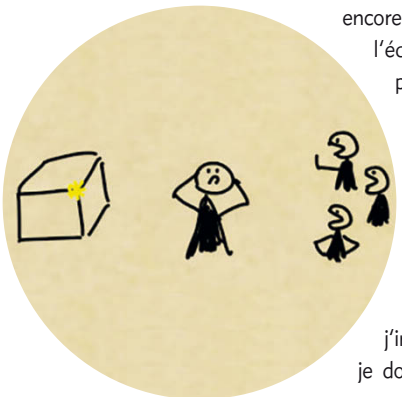
personne ne voulait prendre ce rôle en charge, tout le monde traînait des pieds. Alors, un peu comme quand personne ne veut prendre en charge le support niveau 2 du suivi de production, on décide de faire tourner le rôle de Scrum Master. Spoiler alert : ça ne marche pas. C'est comme demander à un boulanger d'assurer 2 jours sur 6 la fabrication des pâtisseries de sa boulangerie. Assurer une nouvelle spécialité est tout à fait possible, à condition d'être volontaire et motivé. Sinon, vous finissez par vendre des pâtisseries déjà toutes faites !

La seconde situation fait apparaître une réelle envie de découvrir le rôle pour plusieurs personnes de l'équipe. Là, c'est différent : il y a de l'intérêt pour le rôle, et une envie d'apprendre. Cela peut marcher à condition de bien spécifier la priorité des rôles : si je suis à la fois développeuse back et Scrum Master pendant ma semaine d'itération, est-ce qu'on est d'accord que je privilégie ma casquette facilitante au service de l'équipe plutôt que la correction d'un bug de la semaine dernière ? Car oui, cette situation peut très souvent arriver et il faut rapidement statuer sur le comportement à suivre pour ne pas se perdre.



Et le Product Owner ? il connaît l'agilité !

Oui, mais c'est une fausse bonne idée dans le temps. Rappelez-vous le Scrum Guide : « The Product Owner is responsible for maximizing the value of the product resulting from work of the Development Team. » versus « The Scrum Master is a servant-leader for the Scrum Team ». Quand le premier est focalisé sur l'amélioration continue de son produit, le second veille à l'amélioration continue de l'équipe et de ses bonnes pratiques. Ce n'est pas toujours antinomique, mais lorsque la décision de choisir entre donner encore plus pour le produit et protéger l'équipe des demandes inutilement pressantes, c'est toujours bien de pouvoir compter sur deux personnes différentes pour arriver à des compromis acceptables et pour l'équipe et pour le produit.



Quand le manager s'en mêle

Dans les entreprises dans lesquelles j'interviens et dans les formations que je donne, j'aime présenter le manager comme un facilitateur. S'il est rarement LE facilitateur interne à l'équipe, il est celui vers qui j'aime me tourner quand les axes d'amélioration identifiés prioritaires ne peuvent pas venir de l'équipe elle-même (l'auto-organisation c'est bien, mais tous les leviers ne se trouvent pas au travers des compétences de l'équipe). Quand les blocus se situent entre équipes qui ne se parlent pas ou plus, quand il est nécessaire de détendre les cordons de la bourse pour investir dans une solution ou une compétence, quand la connaissance historique de l'équipe est un atout : le manager est souvent la personne clé sur qui compter. J'ai d'ailleurs moi-même un rôle de formation et de coaching du manager dans cette aide apportée aux équipes.

Je n'ai, en revanche, jamais croisé le cas où le manager était le scrum master de l'équipe. Je serai très intéressée par des retours d'expériences dans cette situation. Si le Scrum Guide ne mentionne pas le manager, il ne l'efface pas pour autant. Mais je connais un autre cadre agile qui, par sa description précise de toutes les « cases » de l'entreprise dans son passage à l'agilité à l'échelle, efface littéralement le rôle de manager. Vous voyez duquel je parle ? Oui : SAFe.

Je me souviens d'avoir lu dans un rapport du Cigref de 2018 (*) que « La diffusion de l'agilité à l'échelle de l'entreprise est stimulée par […] le changement de positionnement du manager qui devient un leader-serviteur ». Tiens, tiens, exactement le rôle associé au scrum master dans le Scrum Guide (« The Scrum Master is a servant-leader for the Scrum Team. »). Alors que penser d'un manager qui devient Coach Agile d'un train SAFe dans lequel se trouve l'équipe dont il reste manager sur le papier ? Pour moi, cela casse toute légitimité à le « former et coacher dans son rôle auprès des équipes ». Sa double casquette manager / coach de train, peut le positionner tantôt comme aidant, facilitant, guidant les équipes pour trouver leurs propres solutions, tantôt comme commanditaire ou juge.

Alors si je pouvais faire un souhait pour toutes les équipes de réalisation (de développement ou non), c'est que le manager devrait aider le scrum master de l'équipe, mais il ne peut devenir l'unique référent facilitateur de l'équipe, au risque de faire preuve d'injonctions contradictoires.

Alors, un scrum Master dédié ?

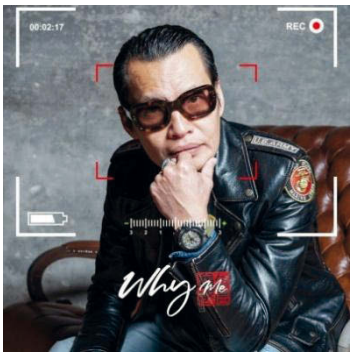
Au tout début de l'histoire d'une équipe, c'est préférable. On découvre les concepts, leurs intérêts, leur possible mise en place dans l'entreprise où l'on se trouve, on est peut-être même la première équipe fonctionnant en mode agile : il faut un agile team advocate qui saura rassurer l'équipe, lui donner une légitimité et la faire naviguer au sein de l'entreprise.

Par la suite, si l'entreprise prend goût à l'agilité, faire multiplier les équipes agiles peut passer par le « partage » d'un Scrum Master sur deux équipes, ou bien en faisant monter en compétence sur ce rôle un équipier de la première heure, pour passer à d'autres équipes à accompagner.

Dans tous les cas, parler aux équipes et prendre en compte leurs retours sur ce rôle est primordial. Et je vous le donne en mille : il existe un moment parfait pour échanger sur le sujet, qui a lieu régulièrement et qui a pour but de faire grandir l'équipe agile… Vous voyez ? la rétrospective !

*https://www.cigref.fr/wp/wp-content/uploads/2018/12/Cigref-Agile-at-scale-Mise-en-oeuvre-agilite-echelle-entreprise-2018.pdf

Le monde de l'art et les NFT



*Charly Ho : https://charlyho.com/

Il y a peu de temps, un de mes amis, artiste - touche à tout, m'a fait part de son projet de création artistique, s'appuyant sur les NFT. Chaque artiste est à la recherche de notoriété, de nouvelles expériences et d'inspiration. CharlyHo* fait partie de ces artistes. Il me donne aujourd'hui l'opportunité de l'aider dans son projet, et j'écris ces lignes, pour lui apporter mes conseils.

Les NFT

Derrière cet acronyme se cache un concept puissant, une révolution est à nos portes. **Non Fungible Tokens** : des jetons informatiques infalsifiables, qui s'appuient sur une technologie cryptographique : **la Blockchain**.

La blockchain, en résumé, est une technologie informatique de base de données décentralisée, dont chaque transaction est tracée, publique et infalsifiable. Les **Blockchains[1]** les plus connues sont actuellement le **Bitcoin (BTC)** et l'**Ethereum (ETH)**. Dans cet article, nous parlons des NFT adossés à l'Ethereum.

Un NFT généré depuis la blockchain **Ethereum**, est un actif numérique, ou un ***Artifact*** numérique. Ce jeton est unique, il est enregistré sur une blockchain et peut-être échangé, comme n'importe quel jeton de la blockchain. Ce qui fait sa valeur, c'est son unicité : *La blockchain du fait de sa nature permet de créer ce type de jeton.*

C'est un peu comme les pièces de monnaie : certaines pièces « particulières » avec un défaut ou une erreur de frappe, s'arrachent à prix d'or auprès des collectionneurs.

La technologie NFT permet de créer des Artefacts uniques, d'identifier leurs auteurs, et de retracer leurs histoires.

Les étapes dans la génération de NFT

Les NFT sont donc des jetons de la **Blockchain Ethereum**. Et comme toute création sur la blockchain, pour en créer ou en acheter, il faut être capable de réaliser des transactions sur celle-ci.

Le Wallet Crypto

Un portefeuille de devises Blockchain représente votre portefeuille numérique. Il vous permet de créer un compte sur la Blockchain choisie.

Ce compte, dès son enregistrement, peut recevoir des crypto-monnaies. Il est très difficile de retenir par cœur son numéro de compte, la plupart des plateformes sur internet vous facilitent sa gestion.

Le point essentiel à retenir, c'est que vos identifiants ne **doivent jamais être perdus**, sous peine de voir vos Crypto bloquées à jamais sur la blockchain.

Penser à les garder à l'abri des regards, et à ne pas les perdre. La blockchain n'a pas de système de « récupération des identifiants ». Vous créez votre portefeuille, vous recevez un mot de passe, et c'est fini. Personne ne pourra changer ces informations, et, si vous ne les communiquez à personne, personne ne pourra les trouver.

Je conseille ces 2 Wallets « Mobile » :

- https://www.coinbase.com/wallet
- https://metamask.io/

Le Courtier Crypto

Ce concept est bien connu des traders qui souhaitent investir en bourse.

La première étape consiste à trouver son « Broker », il vous permettra de réaliser des transactions sur la place de marché de votre choix : Vous lui donnez des ordres, payez des frais, échangez des actions ou des devises.

Les Broker Crypto fonctionnent sur le même principe. Les plateformes suivantes sont simple d'accès :

- https://www.etoro.com/fr/
- https://www.coinbase.com/fr/
- https://robinhood.com/us/en/about/crypto/

Attention, 3 choses à retenir :

- Pensez à transférer vos cryptos régulièrement sur votre portefeuille, sans cela vous ne pourrez pas créer de NFT. Le compte, que votre broker met à votre disposition, est réservé au trading.
- Vos cryptos ne sont pas « physiquement » sur votre portefeuille crypto, ils sont sur la blockchain, votre portefeuille vous permet d'y accéder (tout comme votre carte bleue : votre argent n'est pas dessus, il est à la banque…)
- Vos cryptos, sont votre matière première pour les NFT : ils vous permettent de « payer » la création ou l'achat de NFT.

La plateforme de génération du NFT

La dernière étape dans la génération de NFT est le choix de la plateforme.

Un NFT est un jeton sur la blockchain. Mais créer un NFT est difficile. Les artistes, les designers, les créateurs de contenus et même les experts informatiques, dans leur écrasante majorité, ne seront pas capables de générer un NFT sur une blockchain. Des plateformes ont vu le jour pour remplir ce rôle.

Lorsque vous avez créé votre projet, votre Artifact, il vous faudra le « tokeniser », le transformer en un NFT, que vous pourrez ensuite enregistrer sur la blockchain.

Votre projet peut être une invitation à un événement, une expé-

Julien LEBORGNE

Ingénieur étude et développement, et Passionné d'informatique :)

Très jeune, j'ai découvert la programmation : Mes premières lignes de code étaient en GW-Basic !

J'ai eu la chance de commencer ma carrière professionnelle, en tant que programmeur sur les outils Micro$oft.

Et, depuis près de 20 ans, je participe au sein d'équipes de développeurs, à des projets informatiques, à concevoir des Web-Apps, et à réaliser des missions de maintenance évolutive d'applications existantes.

J'ai rejoint les équipes de Devoteam à l'été 2021, et nous collaborons à créer un environnement motivant pour nos équipes, et à apporter conseil et expertise technique à nos clients.

devoteam
Creative Tech

rience gustative, un voyage offert au possesseur de votre NFT. Vous pouvez également créer des gifs animés, des images 3D, ou de courtes vidéos…

En fait tout « objet » numérique peut être transformé en NFT, et tout objet physique peut être associé à un NFT. Le « contrat » entre l'acheteur d'un NFT et son créateur, est enregistré sur la blockchain et ne peut être falsifié.

Un exemple parlant est celui-ci : Lorsqu'un vendeur s'engage à vous envoyer une montre **Breitling[2]** signée, à l'achat d'un NFT, il est « contraint » de vous l'envoyer, sous peine de voir la confiance de ses futurs acheteurs disparaître.

Les plateformes NFT vous permettent de tokeniser votre création :

- Votre fichier est envoyé sur la plateforme de votre choix. La plupart d'entre elles, vous permettent d'enregistrer des fichiers de 10 Mo à 40 Mo.
- Votre NFT possède de nombreux attributs standard : Nom de l'auteur, date de création, email du créateur, le type d'objet, et du contenu « **unlockable** » (entendez un contenu disponible uniquement pour le possesseur du NFT).

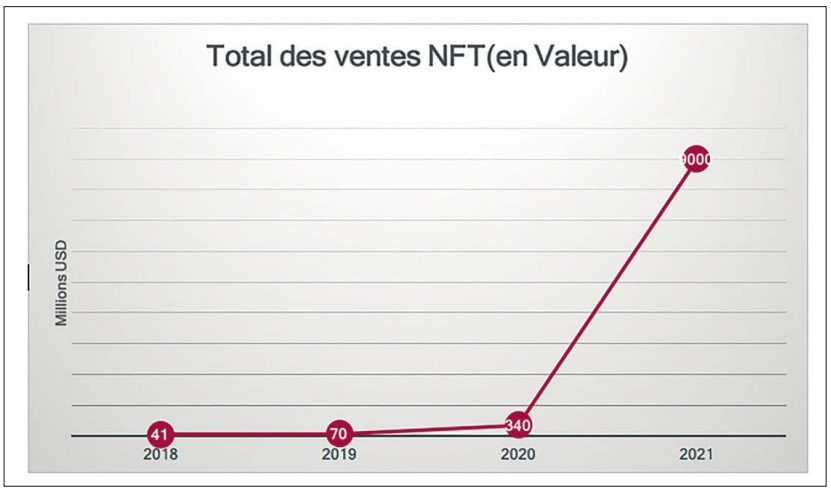
Quelques plateformes connues :

- https://opensea.io/
- https://rarible.com/
- https://foundation.app/
- https://superrare.com/
- https://www.veve.me/

Le marché des NFT

Quelques chiffres

- En 2018 : Total des ventes de NFT - 41 millions USD.
- En 2019 : Total des ventes de NFT – 70 millions USD.
- En 2020 : Total des ventes de NFT – 340 millions USD.
- En 2021 : Total des ventes de NFT - Plus de 9 milliards USD.
- 40 % des ventes mondiales sont réalisées en Chine en 2021.
- Le marché des NFT représente 15,8 milliards de dollars, soit 1,2 % de la capitalisation totale des cryptos (en 2021).



Meilleures ventes de 2021

- CryptoPunk #5217 : 5,59 millions USD.
- Save Thousands of Lives : 5,23 millions USD.
- Stay Free / Edward Snowden : 5,4 millions USD.
- CryptoPunk #5217 : 5,59 millions USD.
- Beeple's Ocean Front : 6 millions USD.
- Beeple's Crossroad : 6,6 millions USD.
- Everydays : the First 5000 Days – 69,3 millions USD.

Sources :

https://www.lesechos.fr/industrie-services/services-conseils/quatre-chiffres-pour-comprendre-comment-les-nft-dopent-le-marche-de-lart-contemporain-1351871

https://thepressfree.com/les-ventes-de-nft-pourraient-atteindre-177-milliards-de-dollars-dici-la-fin-de-2021/

https://fr.cryptonews.com/news/le-marche-des-jetons-non-fongibles-nft-a-triple-en-2020-etud-9347.htm

Conclusion

Les NFT sont à la croisée de plusieurs mondes : Cryptographie, artistique, expert informatique, geek, Start-up, finance…

Pour les artistes, cette technologie permet la monétisation des œuvres d'art numériques. Ils peuvent percevoir de meilleurs revenus : ils peuvent même recevoir une redevance, chaque fois qu'une de leur création est revendue (sur le marché secondaire).

La plateforme Opensea permet d'acheter des collections de Gif, de cartes à collectionner, de tableaux virtuels, de peinture, de lopins de terre virtuels, d'appartements !

Tout peut être transformé en Token ; la pierre angulaire de ce marché est la confiance : confiance dans la robustesse de la **blockchain**, dans la transparence des transactions et des contrats, confiance entre les artistes et leurs mécènes.

Pour conclure, les NFT présentent un double avantage : tirer un profit, une reconnaissance de son travail, mais également un intérêt spéculatif.

Les transactions sont réalisées en Ethereum, cela permet de valoriser son « portefeuille de cryptomonnaies » et de revendre ses ETH au meilleur cours.

Attention cependant : Les NFT doivent être portés par un projet solide, dont l'objectif de rentabilité doit être déconnecté de la valeur du cours.

L'aspect spéculatif ne doit pas prendre le pas sur la viabilité du projet.

Sur le marché 10 % des traders actifs réalisent 85 % des transactions et concentrent **97 % des actifs numériques**. Il vaut mieux se renseigner avant de se lancer !

Pour aller plus loin

Passons à la pratique !

Par un après-midi de janvier, Charly et moi créons le 1er NFT de sa galerie.

Un café noir et sucré, un Wallet (Coinbase dans notre cas), et nous nous connectons à la plateforme Opensea - https://opensea.io/. **Figure 1**

La création d'un **Token** NFT n'est pas difficile :

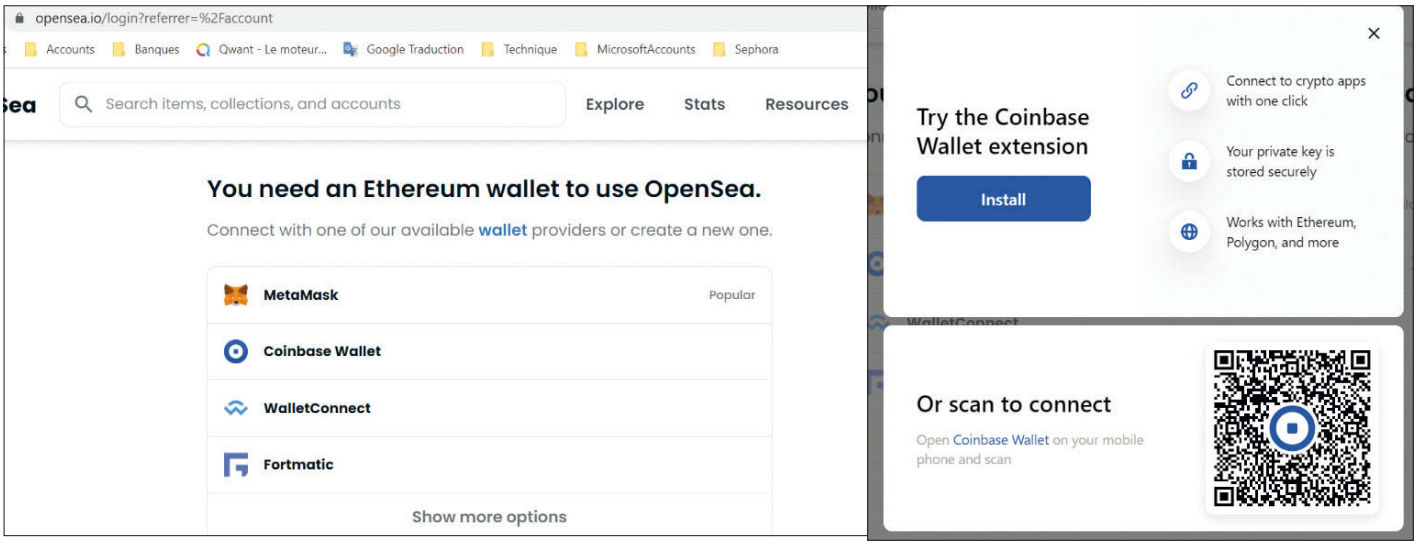
la plateforme vous guide, les actions sont simples, 3 clics suffisent pour accomplir notre but. **Figure 2**

Préparer votre Œuvre

Ce qui prend du temps c'est l'acte de création :

Choisir l'œuvre à transformer en Token. Après quelques débats, nous décidons que les tableaux **Goldorak**, remplissent le mieux les critères de son projet : des tableaux encadrés de belles dimensions (100 x 130), les certificats d'authenticité sont prêts,

Figure 1



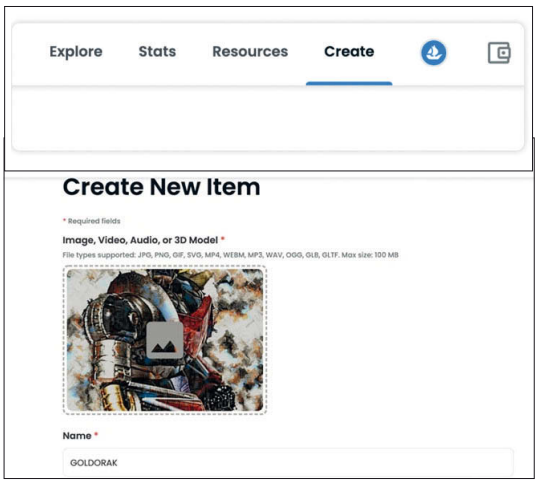


Figure 2

l'œuvre a un caractère nostalgique et un peu Geek.
Les NFT portent de nombreuses informations que l'auteur peut fournir.
Ces attributs associés à notre Token sont précieux, ils construisent un lien entre le collectionneur et l'artiste.
Et voilà après avoir passé les étapes, notre Token est disponible !
Figure 3

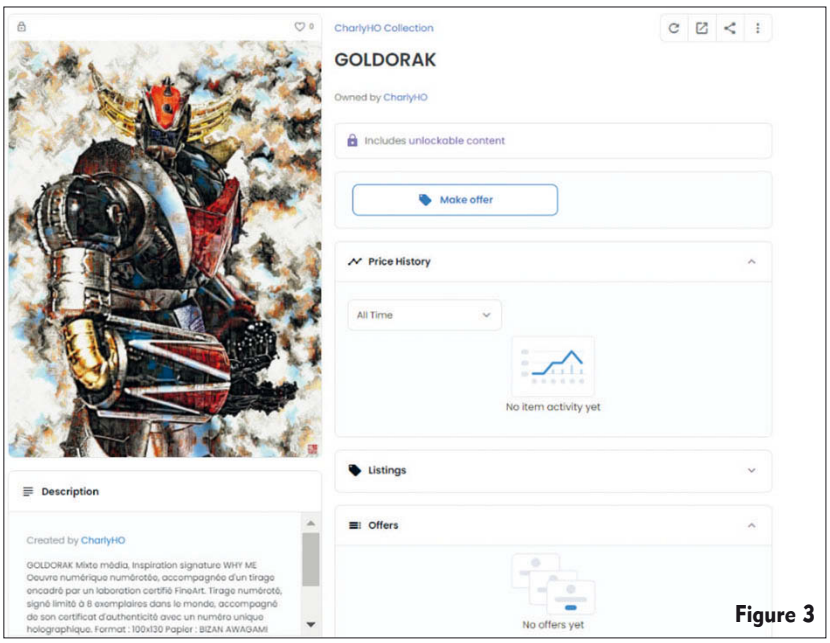


Figure 3

[1] http://www2.assemblee-nationale.fr/static/15/commissions/CFinances/blockchain-synthese.pdf
[2] https://www.breitling.com

Pierre Germain-Lacour

Ancien responsable du calcul scientifique et ancien animateur de la recherche en informatique à PSA Peugeot Citroën.

Opera : un langage et un interpréteur pour effectuer des opérations arithmétiques mathématiquement exactes

Dans les applications informatiques habituelles, on utilise les nombres entiers et les flottants natifs qui sont de tailles fixes. Pour cette raison, la précision des calculs est limitée. En pratique, les approximations qui en résultent sont admises et suffisantes pour des calculs ordinaires.

Pour effectuer des opérations arithmétiques exactes, les nombres utilisés par Opera sont représentés et mémorisés en tant que nombres rationnels, en fractions réduites, avec pour numérateurs et dénominateurs des entiers de tailles aussi grandes que nécessaires. Ce logiciel exécute les sept opérations suivantes à deux opérandes : élever à la puissance, diviser, multiplier, soustraire, additionner et comparer avec cet ordre spécifique de priorités. De plus, il reconnaît aussi automatiquement, d'après le contexte de son emploi, l'opérateur "-" ayant un seul opérande pour utiliser l'opposé du nombre désigné. Pour une expression arithmétique plus complexe, l'emploi des parenthèses est reconnu et effectué. Les exponentiations qui sont irrationnelles sont détectées, approximées et signalées et celles qui sont impossibles sont signalées.

Le nom Opera a pour origine une abréviation qui signifie : opérations arithmétiques mathématiquement exactes.

Opera a deux modes de fonctionnement : l'exécution d'un fichier de commandes et le mode conversationnel, ou interactif. L'interpréteur, dans les deux modes d'utilisation, effectue les calculs écrits en langage Opera. Il est disponible pour Windows et Linux et fonctionne dans une fenêtre en mode console.

Les bases du langage sont :

- Les constantes admises du langage qui sont soit des nombres entiers soit des nombres décimaux.
- Chaque variable représente un nombre rationnel ayant pour valeur une fraction réduite avec dénominateur positif. Les nombres entiers positifs ou négatifs sont reconnus : leurs dénominateurs sont égaux à 1.
- Toutes les variables sont du même type. Elles sont toutes locales dans le fichier de commandes où elles sont utilisées. Il n'existe pas de variables globales, ce qui facilite l'emploi des fichiers de commandes.
- Les variables qui sont des données ou des résultats en arguments d'un fichier de commandes sont indiquées et utilisées pour cela. La transmission se fait par copie.
- Un indice calculé avec un entier, exemple x[n], permet d'utiliser un tableau de variables.
- Un fichier de commandes peut en appeler un autre.
- Les branchements et débranchements conditionnels sont possibles.
- Les boucles itératives sont possibles, une boucle peut en contenir d'autres.
- On peut lire dans un fichier une suite de variables. On peut aussi sauvegarder toutes variables connues.
- Le caractère # débute un commentaire dans les fichiers de commandes.
- La variable last contient une copie du dernier résultat obtenu précédemment à son utilisation.
- Les fonctions reste, pgcd, ppcm, hasard, invmod, expmod, pnps, facteur et prem sont spécifiquement utilisables avec des variables ou des expressions ayant pour valeurs des entiers.

Un premier exemple

Voici un exemple simple d'utilisation de l'interpréteur Opera. Soit le fichier verif.txt suivant.

```
n = 366482613450380223856974383038858330841709595616663701388948644 73
p = 4103188409
q = 893165453106007309002842289513930850163184879519052329 7
prem p
prem q
v = p*q
valeur n-v
```

Si on lance Opera en mode conversationnel et si on écrit la commande : exec verif.txt, on obtient un affichage qui montre que les nombres p et q sont premiers et que n et v sont égaux. En chiffrement RSA si on connaît la clef publique d'un auteur pour découvrir sa clef privée il faut trouver les deux facteurs premiers d'un nombre entier ayant plusieurs centaines de chiffres décimaux. Opera n'a pas la possibilité de faire une telle décomposition, mais il peut très facilement en effectuer la vérification, si on le souhaite, c'est ce que fait cet exemple. En supplément, le fichier de commandes rsa.txt disponible dans la publication de Opera est un prototype de chiffrement RSA pour en montrer son utilisation dans l'envoi d'un document chiffré.

Les instructions du langage Opera

Les sept opérateurs binaires ont pour symboles : ^ / * - + < > et ils ont cet ordre de priorités. La division par 0 est possible. L'instruction i = 1/0 définit l'infini positif et j = -1/0 définit l'infini négatif. Voici deux expressions valides : 1/2*3/4 résultat : 3/8 et : 3*(8-2)^2 résultat : 108. L'ordre des priorités des opérateurs est important pour écrire, et pour lire, les expressions arithmétiques. Si l'expression comporte plusieurs fois le même opérateur, le calcul est effectué de gauche vers la droite. À titre d'exemples, on peut noter que : 1+3*4 = 3*4+1 = 13 et : 3^3^3 = 27^3 = 19683 et non pas 3^27 = 7625597484987. L'utilisation des parenthèses permet de modifier l'ordre des calculs effectués : (1+3)*4 = 16 et : 3*(4+1) = 15.

L'exponentiation, exemple : x = a^n, a un fonctionnement particulier. En effet, si n = p/q est un nombre rationnel avec q > 1 il y a des cas où le résultat x est un nombre rationnel et d'autres cas où il est irrationnel, exemples : (4/9)^(3/2) = 8/27 et 2^(1/2) est irrationnel. Si le résultat est irrationnel, Opera calcule une approximation du résultat et affiche une indication de cette exception. Il y a une fonction qui permet de choisir le niveau de précision de cette approximation. De plus, quand le résultat n'est pas un nombre réel, exemple : (-1)^(1/2), un message signale ce cas éventuel. Enfin, l'utilisation sans précaution de l'exponentiation peut provoquer un encombrement de la mémoire centrale quand n et a sont très grands, ce qu'il faut éviter.

Le manuel de l'utilisateur explique les 33 instructions du langage parmi lesquelles on trouve, par exemple, les quatre suivantes. La commande "exec" (exemple : exec fichier.txt a, b, c) pour indiquer en existence et valeur les données et en existence les résultats d'un fichier de commandes que l'on souhaite lancer. La commande "boucle" notifie l'instruction qui la précède puis si last est positif continue en séquence et si non elle va chercher l'instruction qui suit l'instruction "retour" qui lui correspond. On peut imbriquer une "boucle" dans une autre. La commande "facteur" (exemple : facteur n) calcule un facteur premier d'une variable ou d'une expression entière qui a 16 ou 17 chiffres décimaux au maximum pour être performant. La commande "copier" (exemple : copier n,r) est nécessairement la première instruction effective d'un fichier de commandes. En complément de la commande "exec" elle désigne les variables en données et les variables en résultats de son fichier de commandes et la transmission se fait par copie.

La programmation en C++ de l'interpréteur Opera

Le code source de Opera est programmé en C++ et n'a qu'une seule dépendance : la bibliothèque GMP qui est disponible avec les autres fichiers *.cpp. Opera ne met rien dans le système d'exploitation, il n'a pas d'installeur, il est portable, on peut l'utiliser dans une clef USB.

Trois classes C++ particulières sont importantes. La classe bigRa effectue le traitement arithmétique des nombres rationnels ayant la taille aussi grande que nécessaire. La classe listera est une liste doublement chaînée servant à gérer les variables utilisées pendant l'exécution du programme Opera. Quand un fichier de commandes en appelle un autre, il faut conserver les variables du fichier appelant pendant l'exécution du fichier appelé. Pour cette raison, le nom des variables gérées par la classe listera comporte un préfixe défini par le niveau d'exécution du fichier de commandes en cours, ce qui assure l'utilisation locale de ses variables internes. Et la classe Trans gère par copie la transmission des variables entre le fichier de commandes appelant et le fichier de commandes appelé. Quand le fichier appelé est terminé, ses variables sont supprimées.

Pour interpréter une expression arithmétique, il est habituel d'utiliser l'intermédiaire de la notation polonaise inversée. Opera ne l'utilise pas, il effectue directement le calcul du résultat. Quand l'expression comporte des parenthèses, il en vérifie l'équilibre et ensuite il remplace le contenu des parenthèses les plus internes par une variable temporaire calculée pour cela. Ceci permet, en utilisant la récursivité du C++, d'évaluer seulement des expressions sans parenthèse : pour 3*(8-2)^2 on calcule a = 8-2 puis 3*a^2. En fait, les noms de ces variables intermédiaires sont obtenus automatiquement, exemples : &t1, &t2..., et ne peuvent pas être en conflit de nom avec les variables de l'utilisateur.

Pour évaluer la valeur d'une expression sans parenthèse, il faut respecter la priorité des opérateurs utilisés. Pour l'opérateur - ayant un seul opérande, il y a deux traitements différents selon les cas d'utilisation. Dans le premier cas, exemple : -123 ou -v, on effectue le changement : 0-123 ou 0-v. Dans le deuxième cas, exemple : 3*-v on effectue le changement temporaire 3*(-v) qui deviendra ensuite 3*t en raison du traitement des parenthèses. Après cela, l'expression finale a seulement des opérateurs ayant deux opérandes. Si on a plusieurs fois le même opérateur dans une expression, le calcul est effectué de gauche à droite.

D'autres exemples du langage Opera

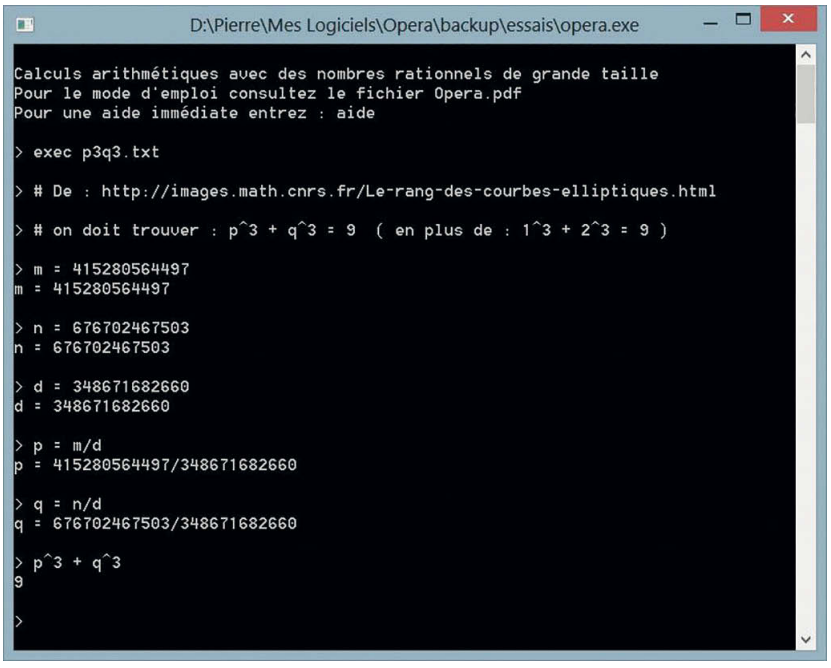
Voici un programme en langage Opera qui vérifie la démonstration de Zagier qui prouve que le nombre 157 est congruent.

```
# Ce fichier de commandes montre que 157 est congruent.
# Un nombre entier n est congruent s'il existe 3 nombres
# rationnels a, b et c tels que a^2 + b^2 = c^2 et n = ab/2
# Voir : https://fr.wikipedia.org/wiki/Nombre_congruent
# Zagier a démontré que 157 est congruent.
# Voir : http://serge.mehl.free.fr/chrono/Zagier.html
# Le triplet (a, b, c) approprié pour n = 157 est :
a = 6803298487826435051217540/411340519227716149383203
b = 41134051922771614938320 3/216665556937147613096 10
n = 224403517704336969924557513090674863160948472041
d = 8912332268928859588025535178967163570016480830
c = n/d
a2 = a*a
b2 = b*b
c2 = c*c
d2 = (a2 + b2) - c2
n = (a*b)/2
# Comme prévu on obtient : d2 = 0 et n = 157
```

Voici un programme écrit en langage Opera qui calcule une suite d'approximations du nombre pi avec une formule due à Jean-Henri Lambert jusqu'à obtenir n décimales exactes.

```
# Ce programme en langage Opera calcule une suite d'approximations de pi
# jusqu'à celle ayant c décimales exactes avec la formule suivante.
# pi = 4 / (1 + 1^2 / (3 + 2^2 / (5 + 3^2 / (7 + 4^2 / (9 +... )))))
# Pour l'utiliser, on peut écrire :
# n = 16
# exec mpi.txt n
# Ensuite on peut lire le fichier mpi-out.txt obtenu pour afficher
# toutes les variables utilisées où la dernière a la précision demandée.
# Mais on peut aussi écrire seulement :
# lire mpi-out.txt
# valeur f[i]
# Et on vérifie que l'approximation finale a 16 décimales exactes.
copier c
f[1] = 3
f[2] = 19/6
delta = f[2]-f[1]
i=2
delta > f[i]/10^c
boucle
 i = i + 1
 k = i
 d = (2*k-1)+k^2/(2*k+1)
 k > 1
 boucle
   k = k-1
   d = (2*k-1)+k^2/d
 retour
 f[i] = 4/d
 delta = f[i]-f[i-1]
 si delta < 0
   delta = -delta
retour
enti f[i]*10^c
mpi = last
garder mpi-out.txt
```

Figure 1



f[22] = 51359350521741312/16348189019048887 est l'approximation finale du nombre pi obtenue avec le programme mpi.txt présenté ci-dessus. Pour 32 décimales exactes on aurait obtenu :
f[43] = 377007404451960464990387692952593891328/120005184001549872126533838222292079175

Voici une image de la console Windows avec un petit exemple d'utilisation du langage. **Figure 1**

Utilisations du langage

Opera est public et gratuit. La publication comporte la documentation avec un manuel de l'utilisateur, le code source pour Windows et pour Linux, des commandes de compilation du code source, l'exécutable pour Windows et celui pour Linux et de nombreux exemples d'utilisation.

Opera est disponible à https://github.com/pgl10/Opera. Son exécutable pour Windows est obtenu avec Visual Studio et celui pour Linux est obtenu avec gcc. On peut aussi compiler le code source avec d'autres compilateurs C++.

Il faut noter que les espaces dans les instructions Opera ne sont pas effectifs sauf, bien sûr, dans les commentaires. Ce qui permet d'écrire aussi bien : n = 3 + 4*5 que : n=3+4*5. En effet, au début de l'interprétation de l'instruction les espaces sont supprimés, sauf dans les commentaires. Dans un fichier de commandes, l'indentation des instructions n'est pas indispensable, mais c'est une disposition commode. De plus, pour lire et effectuer la dernière ligne effective d'un fichier de commandes il est nécessaire d'ajouter ensuite une ligne vide. Les exemples ci-joints montrent cette nécessité.

Opera a trois instructions pour effectuer des entrées ou des sorties. La commande "noter" (exemple : noter fichier.txt) permet de noter dans un fichier toutes les variables actuelles. Si le fichier désigné existe déjà, l'archivage est refusé. La commande "garder" (exemple : garder fichier.txt) permet de garder dans un fichier toutes les variables actuelles. Si le fichier désigné existe déjà, il est remplacé. La commande "lire" (exemple : lire fichier.txt) permet de créer au niveau actuel les variables définies dans ce fichier, ce qui est très commode quand on souhaite faire divers essais avec les mêmes nombres.

Opera a divers concurrents. Le premier est le C++ avec la bibliothèque GMP où se trouvent les entités mpq_t. Il y a aussi les divers langages formels, exemples : Maple, MatLab ou Eigenmath. Opera est moins puissant, mais dans bien des cas il est plus commode. Et le code source d'Opera comporte des éléments pouvant servir à d'autres développements analogues.

Louons en web 3.0 avec notre Dapp!

Les progrès récents et rapides du langage Solidity associés à l'apparition d'outils open-source comme « truffle » « ganache » permettent de créer un nouveau type d'application qui fonctionne sur un réseau distribué communément appelé « decentralized application » ou dapp. L'application n'est plus hébergée sur un serveur centralisé, mais sur un réseau décentralisé peer-to-peer.

Thierry Fondrat

Professeur agrégé, enseignant l'informatique depuis de nombreuses années, j'ai rédigé ce tutoriel sur la création d'un contrat intelligent en web 3.0, depuis la conception jusqu'à l'interface graphique utilisateur

contact :
solsecu974@gmail.com

Cet article va présenter les étapes de développement d'une application de location d'appartements décentralisée s'appuyant sur Ethereum. Bien sûr, il ne s'agira pas de véritables appartements, mais d'images de ces appartements ! En l'occurrence de jetons non fongibles (NFT, « non-fungible token »). Cet article s'adresse à ceux qui ont des notions sur les blockchains et leurs applications, mais voudraient en avoir une approche concrète.

1) En quoi une DAPP est-elle différente d'une application web classique?

Dans une architecture client-serveur web classique, un navigateur se connecte à un serveur, associé à une base de données bien protégée à l'abri des intrusions. Les données sont donc cachées chez l'hébergeur. Les transactions qui se font entre le client et le serveur sont donc cachées elles aussi.
Dans le réseau Ethereum, les transactions qui se produisent sont empaquetées dans des blocs et chaque bloc est lié au bloc suivant. Cette série de blocs liés qui contient toutes les données de transaction constitue la blockchain.
Ainsi, dans le cas d'une vente, d'un remboursement ou d'un litige, chaque transaction entre acheteurs et vendeurs sera enregistrée sur la blockchain et est accessible à tous. Dans une Dapp, la base de données devient accessible publiquement et en toute sécurité par tout le monde, ce qui permet d'envisager de passer toute sorte de contrat sans avoir besoin d'intermédiaire tout en assurant la fiabilité de la transaction.
Pour s'assurer que tous les nœuds du réseau ont la même copie des données et pour s'assurer qu'aucune donnée invalide n'est écrite dans cette base de données, Ethereum utilise un algorithme appelé « Proof of Work » (également connu sous le nom Ethereum Mining). Cet aspect, qui est très gourmand en ressources et donc catastrophique pour l'environnement, est destiné à disparaître avec la future version 2 d'Ethereum. Kintsugi, le premier testnet permettant à la communauté de tester Ethereum 2.0, est disponible depuis décembre 2021. La date du 22 juin 2022 a été avancée par @Superphiz, le consultant en santé communautaire de la chaîne Beacon d'Ethereum.
Pour comprendre le fonctionnement, prenons par exemple le cas d'un client web. Chaque navigateur client communique avec sa propre instance de l'application. Quiconque qui veut interagir avec un Dapp aura besoin d'une copie complète de la blockchain en cours d'exécution sur son ordinateur, téléphone, etc. **Figure 1**

On pourrait se dire que donc, avant de pouvoir utiliser une application, il faudrait télécharger l'ensemble de la blockchain. Cela présente l'avantage de ne pas s'appuyer sur un serveur central unique qui pourrait disparaître d'un jour à l'autre. Par contre, cela nécessiterait beaucoup de ressources sur votre machine (ram, disque dur).
Il y a quelques moyens de garder l'application décentralisée tout en rendant l'interaction rapide et facile, avec MetaMask par exemple, qui peut s'installer en tant que plugin sur votre navigateur. (https://metamask.io/) METAMASK

2) Les outils :

Le code de l'application permettant d'acheter, vendre ou annuler est appelé « contrat intelligent».
Il s'écrit en langage Solidity. Similaire à JavaScript, Solidity est typé statiquement. Il prend en charge l'héritage, les bibliothèques et les types complexes définis par l'utilisateur, entre autres.
Le compilateur de Solidity, appelé solc, permet de générer de l'Ethereum Byte Code et ensuite de déployer du « bytecode » dans la blockchain. Il y a d'autres alternatives à Solidity mais Solidity est de loin le langage le plus populaire pour le développement de contrats.
L'image illustre un contrat déployé dans l'EVM : **Figure 2**

Donc, en résumé, la blockchain stocke vos données, stocke le code et exécute également le code dans l'EVM (Ethereum Virtual Machine).
Pour construire des applications web dapp, Ethereum est livré avec une bibliothèque javascript pratique appelée **web3.js**

Figure 1

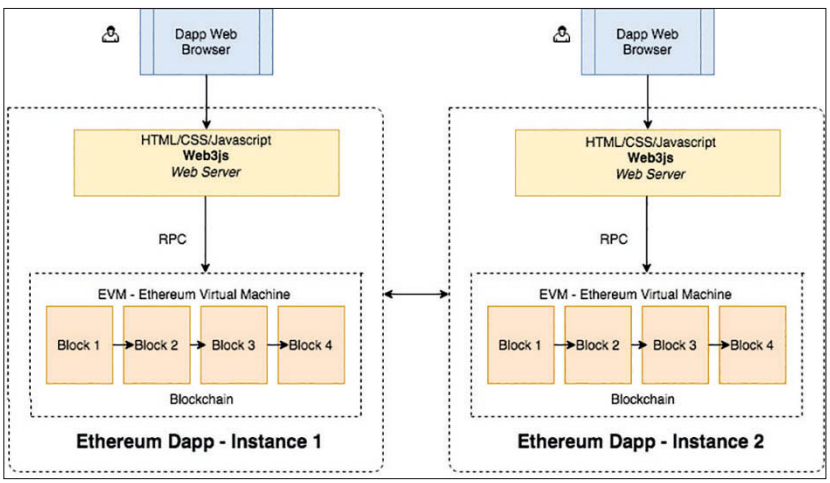


Figure 2

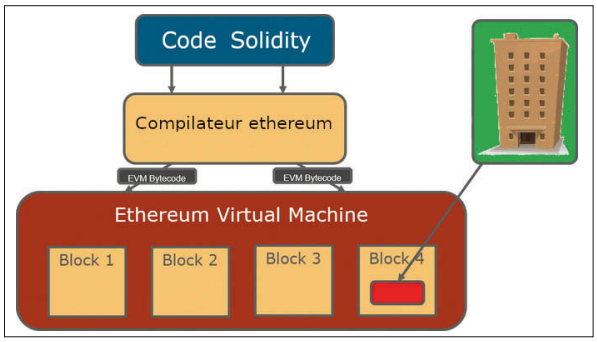


Figure 3

Figure 4

qui se connecte à votre nœud blockchain. Vous pouvez donc simplement inclure cette bibliothèque dans votre framework JS favori (vue, express, reactjs, angularjs etc.) pour construire l'interface de l'application.
Une autre caractéristique importante est l'aspect financier de la plateforme. Dès que vous commencez à utiliser une dapp, vous utilisez un compte bancaire intégré, dans un portefeuille où vous stockez de l'argent (en Ether — la monnaie utilisée dans l'écosystème Ethereum).
Les transactions sont payantes, en un sous multiple très faible de l'éther appelé gas.

Remarque : pour davantage de bases sur ethereum, vous pouvez lire le tutoriel sur trufflesuite. **Figure 3** (https://www.trufflesuite.com/guides/ethereum-overview)

2) Cahier des charges et installation des outils :

Votre agence immobilière dispose de 16 appartements situés dans un immeuble nouvellement construit, que vous voulez mettre en ligne, en utilisant une application décentralisée. Il va falloir créer une dapp qui associe une adresse Ethereum avec chaque appartement. Notre travail consiste à rédiger le contrat intelligent et la logique initiale pour son utilisation. La structure et le style du site Web seront fournis.
Il faut tout d'abord installer le framework opensource truffle. D'après ses auteurs, truffle est "un environnement de développement, un cadre de test et un pipeline d'actifs pour les blockchains utilisant la machine virtuelle Ethereum (EVM), visant à rendre la vie en tant que développeur plus facile".
Truffle s'appuyant sur node.js, il vous faudra installer au préalable node.js et git:
sudo apt-get install node.js (https://nodejs.org/fr/download/)
sudo apt-get install git
Une fois ceci installé, on peut installer truffle :
npm install -g truffle (l'option -g permet une installation globale sur votre système)
Pour plus d'informations : https://www.trufflesuite.com
Pour vérifier que **Truffle** est bien installé, tapez truffe sur un terminal. En cas d'erreur, assurez-vous que vos modules npm sont ajoutés à votre chemin.
Il serait possible de développer votre application directement en réel avec vos chers ethereum, mais il y a une autre solution moins chère :). C'est ici qu'apparaît **Ganache**, une blockchain personnelle pour le développement d'Ethereum conçue pour déployer des contrats, développer des applications et exécuter des tests.
Vous pouvez télécharger Ganache en allant sur http://trufflefra-mework.com/ganache.
Il est aussi possible d'utiliser Truffle Develop, la blockchain personnelle intégrée de Truffle, au lieu de Ganache, pour se passer de l'interface graphique et aller plus vite. Vous devrez dans ce cas modifier certains paramètres notamment le port sur lequel fonctionne la chaîne de blocs pour adapter le tutoriel à Truffle Develop.

3) Création du projet Truffle à l'aide d'une Truffle Box

Il faut tout d'abord créer votre répertoire de travail et se placer dedans :
mkdir agenceLocation
cd agenceLocation
Pour le développement, on peut partir de zéro , utiliser des ID en ligne (https://remix.ethereum.org) ou utiliser des trufflebox. Les trufflebox sont des configurations pré-établies permettant de démarrer très rapidement. Beaucoup sont proposées, avec différents frameworks de front-end comme fluidity, react, vue, express… Vous trouverez ainsi sûrement votre bonheur !
Dans cet article, nous allons nous contenter d'utiliser «pet-shop » et de l'adapter à notre besoin.
Tout d'abord, il faut télécharger la boite:
truffle unbox pet-shop

Remarque : Une autre commande d'initialisation utile est « truffle init », qui crée un projet Truffle vide sans exemple de contrat inclus. Pour de plus amples renseignements, veuillez consulter la documentation sur la création d'un projet.
(https://www.trufflesuite.com/docs/truffle/getting-started/creating-a-project)

Cette boite comprend la structure de base du projet ainsi que le code pour l'interface utilisateur.
Parmi les fichiers de configuration, on trouve truffle-config.js qui permet de modifier les ports. bs-config.json sert pour la configuration du serveur web de test.
Le contenu du dossier est alors le suivant **(Figure 4) :**

contracts/ : Contient les fichiers sources de Solidity pour nos smart contracts. Il y a ici un contrat important appelé Migrations.sol, dont nous parlerons plus tard.
migrations/ : Truffle utilise un système de migration pour gérer les déploiements de contrats intelligents. Une migration est un contrat intelligent spécial supplémentaire qui suit les changements.
test/ : Contient des tests JavaScript et Solidity pour nos contrats intelligents.
src/ : Contient le front-end du site.
node_modules/ : Contient les librairies de node.js utilisées dans le projet.

4) Rédaction et compilation du contrat intelligent

Un contrat est semblable à une classe en langage objet.
On définit donc ici le contrat « Location » dont la donnée principale est un tableau d'adresses appelé appartements.

Comme indiqué précédemment, Solidity est un langage typé statiquement, ce qui signifie que les types de données des contrats doivent être définis, comme les chaînes, les entiers et les tableaux. Les tableaux contiennent un type de données et peuvent avoir une longueur fixe ou variable.
De plus, Solidity a un type unique appelé address. Les "address" sont des adresses Ethereum, stockées sous forme de valeurs de 20 octets. Les adresses peuvent être payables (héritage dans le diagramme ci-dessous), ce qui ne sera pas utilisé ici. Chaque compte et contrat intelligent sur la blockchain Ethereum a une adresse et peut envoyer et recevoir des ether vers et depuis cette adresse.
Le traitement des NFT n'étant pas le sujet ici, nous nous contenterons ici pour nos appartements d'un tableau de 16 adresses Ethereum. **Figure 6**
De plus, les variables publiques ont des méthodes getteur automatiques, un getteur ne serait donc pas nécessaire s'il n'y avait qu'une seule adresse. Cependant, dans le cas d'un tableau, une seule clé est renvoyée vers une seule valeur. L'interface utilisateur doit mettre à jour tous les statuts d'adoption des animaux. C'est pourquoi une fonction getAppartements, qui renvoie le tableau entier, est présente. Puisque les appartements sont déjà déclarés, nous pouvons simplement les retourner. Le type de retour est spécifié comme 16 adresses mémoire. Le mot clé « memory » indique l'emplacement des données pour la variable. Le mot-clé « view » dans la déclaration de fonction signifie que la fonction ne modifiera pas l'état du contrat.
Le diagramme de classe de la partie back-end ressemble à celui de la **Figure 5.**

La fonction loue :
Premier arrivé, premier servi ! Elle permet à celui qui demande de louer un appartement et renvoie l'apartId correspondant.
Dans Solidity, les types de paramètres de fonction et de sortie doivent être spécifiés. Dans ce cas, nous allons prendre un apartId (entier) et renvoyer un entier. L'instruction require() permet de s'assurer que le paramètre apartId est bien compris entre 0 et 15.
Si l'ID est dans la plage, nous ajoutons alors l'adresse qui a fait l'appel à notre tableau appartements.
L'adresse de la personne ou du contrat intelligent qui a appelé cette fonction, est indiquée par msg.sender. Pour un exemple plus évolué, il faudrait bien entendu ajouter un tableau de loueurs au contrat et s'assurer que l'appartement n'est pas loué !
Une fois toutes ces considérations prises en compte, il reste à créer le fichier Location.sol dans le dossier contracts et à le compiler avec : truffle compile **Figure 6**
Cette compilation génère du « **bytecode »** (json) destiné à la machine virtuelle Ethereum (EVM).
Deux fichiers json sont créés : Location.json et Migration.json.
Maintenant que nous avons compilé avec succès nos contrats, il est temps de les migrer vers la blockchain.

5) Migration vers une blockchain locale :

Nous sommes maintenant prêts à créer notre propre script de migration ! Une migration est un script de déploiement destiné à modifier l'état des contrats de votre application, en la déplaçant d'un état à l'autre.
Pour la première migration, vous pouvez simplement déployer du nouveau code, mais avec le temps, d'autres migrations peuvent déplacer des données ou remplacer un contrat par un nouveau.

Note : *En savoir plus sur les migrations dans la documentation Truffle.* (https://www.trufflesuite.com/docs/truffle/getting-started/running-migrations)

En plus du fichier 1_initial_migration.js, ajouter 2_deploy_contracts.js **Figure 7.**
Avant de pouvoir migrer notre contrat vers la blockchain, nous avons besoin d'une blockchain en cours d'exécution.
Pour ce tutoriel, nous allons utiliser Ganache, une blockchain personnelle pour le développement d'Ethereum que vous pouvez utiliser pour déployer des contrats, développer des applications et exécuter des tests. **Figure 8**

Figure 5

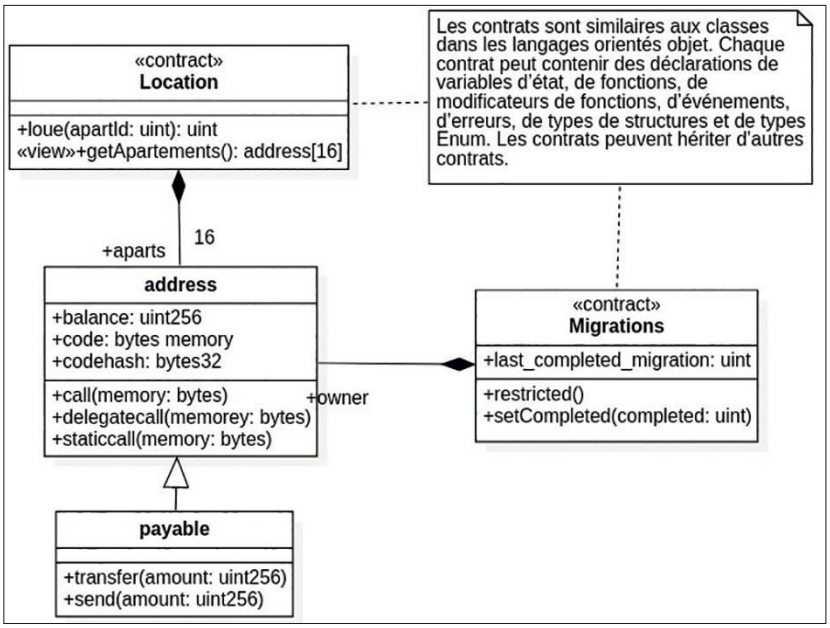


Figure 6

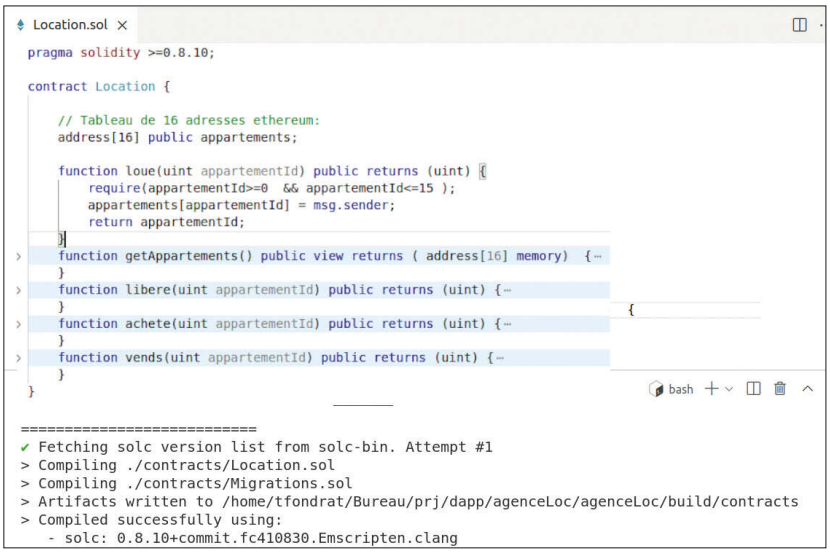


Figure 7

```
migrations > JS 2_deploy_contracts.js > <unknown> > exports
1  var Location = artifacts.require("Location");
2
3  module.exports = function ( deployer ) {
4      deployer.deploy(Location);
5  }
```

Figure 8

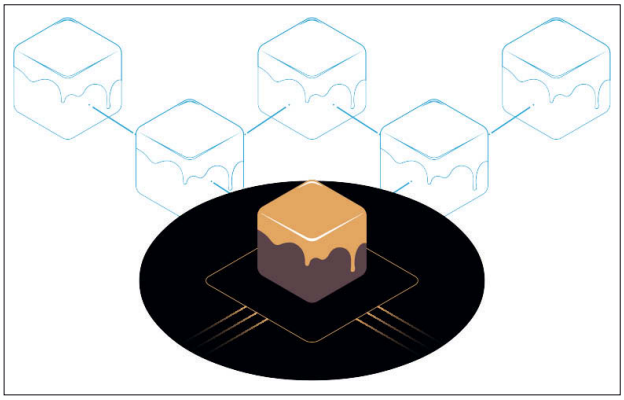
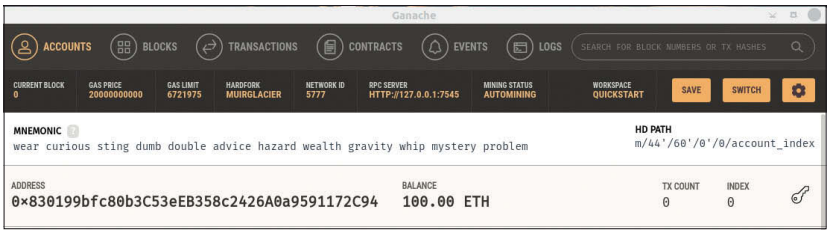


Figure 9



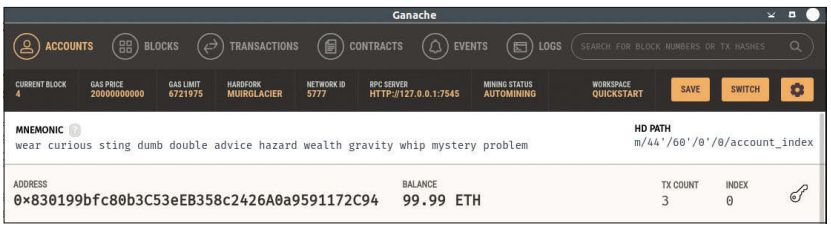


Figure 10

Si ce n'est pas déjà fait, téléchargez Ganache et double-cliquez sur l'icône pour lancer l'application. Cela générera une blockchain fonctionnant localement sur le port 7545.

Documentation sur ganache :
https://www.trufflesuite.com/docs/ganache/using

Lancer l'application, et l'interface suivante apparaît : Figure 9

Ne rêvez pas, vous n'avez pas gagné 100 éthereum ! Ce compte est virtuel…

De retour dans notre terminal, migrer le contrat vers la blockchain en lançant la commande suivante : **truffle migrate ;**

Vous devriez voir une sortie similaire à ce qui suit :

```
1_initial_migration.js
======================

  Deploying 'Migrations'
  ----------------------
  > transaction hash:   0x3b558e9cdf1231d8ffb3445cb2f9fb01de9d0363e0b97a17f9517da318c2e5af
  > Blocks: 0           Seconds: 0
  > contract address:   0x5ccb4dc04600cffA8a67197d5b644ae71856aEE4
  > account:            0x8d9606F90B6CA5D856A9f0867a82a645e2DfFf37
  > balance:            99.99430184
  > gas used:           284908
  > gas price:          20 gwei
  > value sent:         0 ETH
  > total cost:         0.00569816 ETH

  > Saving migration to chain.
  > Saving artifacts
  -------------------------------------
  > Total cost:         0.00569816 ETH

2_deploy_contracts.js
=====================

  Deploying 'Adoption'
  ....................
```

Vous pouvez voir les migrations exécutées dans l'ordre, suivies de quelques informations liées à chaque migration. **Figure 10**

On peut remarquer que l'état de la blockchain a changé dans Ganache. La blockchain montre maintenant que le bloc actuel, auparavant 0, est maintenant 4. En outre, alors que le solde à l'origine était de 100 ether, il est maintenant inférieur, en raison des coûts de transaction de la migration évoqués précédemment.

Vous avez maintenant écrit votre premier "smart contract" et déployé dans une blockchain locale. Il est temps d'interagir avec votre contrat intelligent pour vous assurer qu'il fasse ce qu'on lui demande.

Il est possible de tester votre contrat de manière interactive, ou en utilisant des librairies spécialisées telles que https://www.chaijs.com/ ou mochajs.org, mais pour ne pas alourdir cet article, cette partie ne sera pas traitée ici.

6) Création d'une interface utilisateur pour interagir avec le smart contract:

Maintenant, il est temps de créer une interface graphique pour l'application.

La structure d'une application front end est déjà présente dans la boîte truffle. Elle se trouve dans le répertoire src/.

On va s'occuper des fonctions spécifiques à Ethereum. De cette façon, vous pourrez comprendre et appliquer ceci à votre propre développement de front-end.

Le schéma **Figure 11** illustre le fonctionnement de l'application et peut être facilement réutilisé pour toutes sortes d'applications. Le fichier html référence app.js qui déclare un objet global App.

L'objet global App implanté dans le fichier app.js permet de charger les informations concernant les appartements dans la fonction init(). Cette fonction exécute ensuite de manière asynchrone l'appel à la fonction initWeb3().

L'interaction avec la blockchain Ethereum se fait en utilisant la librairie javascript web3. Celle-ci permet de récupérer des comptes utilisateurs, envoyer des transactions, interagir avec les smarts contracts…

Pour qu'une application web utilisant Ethereum fonctionne, il faudra:

- Détecter le fournisseur (**window.ethereum**)
- Détecter à quel réseau Ethereum l'utilisateur est connecté.
- Récupérer le compte Ethereum de l'utilisateur.

La fonction initWeb3 se charge donc tout d'abord de la

recherche du fournisseur. En effet, dans un navigateur moderne dapp ou pour une version récente de MetaMask, un fournisseur Ethereum est injecté dans l'objet window.
Le code de la fonction initWeb3 est le suivant :

```
// Si un fournisseur ethereum est présent :
if (window.ethereum) {
    App.web3Provider = window.ethereum;
    try {
    // Request account access
    await window.ethereum.request({ method: "eth_requestAccounts" });;
    } catch (error) {
    // User denied account access...
    console.error("User denied account access")
    }
}
// Autre cas de figure :
else if (window.web3) {
    App.web3Provider = window.web3.currentProvider;
}
// Sinon, accès direct par le port TCP de Ganache
else {
App.web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');
}
```

La fonction se termine avec l'appel de la fonction initContract, qui se charge de l'instanciation du smart contract.
Truffle a une librairie à cet effet appelée @truffle/contract, permettant d'extraire les informations à propos du contrat situées dans Location.json, généré lors de la compilation, synchronisé au cours des migrations, ce qui fait que vous n'avez pas besoin de changer manuellement l'adresse du contrat déployé.
La fonction initContract va donc charger un Artifact à partir de ce fichier. Les Artifacts sont des informations à propos de notre contrat, comme son adresse de déploiement et son Application Binary Interface (ABI). L'ABI est un autre objet JavaScript, qui définit comment interagir avec le contrat, en incluant ses variables, fonctions et leurs paramètres.
Ce qui donne le code suivant :

```
$.getJSON('Location.json', function(data) {
// Get the necessary contract artifact file and instantiate it with @truffle/contract
var locationArtifact = data;
App.contracts.Location = TruffleContract(locationArtifact);
```

Une fois le contrat instancié, nous indiquons le fournisseur en utilisant la valeur de App.web3Provider stockée précédemment

```
App.contracts.Location.setProvider(App.web3Provider);
```

Et enfin, l'appel à markLoue() permet d'extraire et de marquer les appartements loués.

```
return App.markLoue();
```

Cette fonction markLoue est destinée à être appelée plusieurs fois, chaque fois qu'un appartement est loué.

Description de la fonction markLoue qui se charge de la récupération des appartements loués et de la mise à jour l'interface utilisateur:
Après avoir accédé au contratLocation, un appel à la fonction getAppartements() est effectué :

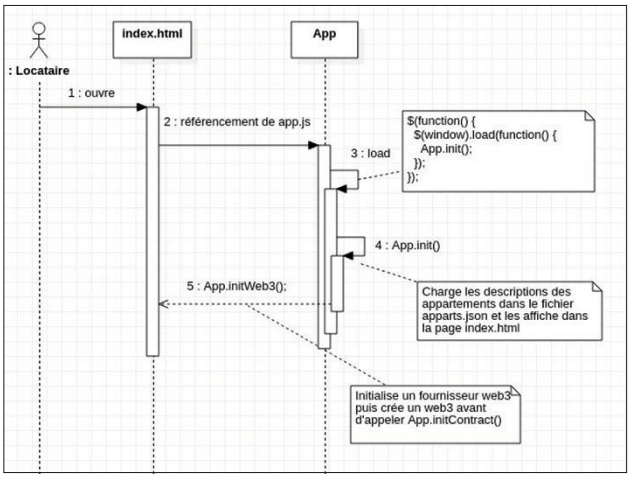


Figure 11

```
var locationInstance;
App.contracts.Location.deployed()
.then(function(instance) {
locationInstance = instance;
// Appel à la fonction getAppartements. Le résultat est envoyé sur la fonction suivante
return adoptionInstance.getAppartements.call();
})
```

L'appel à **call(),** lors du retour de cette fonction, nous permet de lire les données de la blockchain sans avoir besoin d'envoyer une transaction complète, et donc de dépenser des ethers. Après cet appel, une boucle permet de vérifier pour chaque appartement s'il est loué.

```
.then(function(appartements) {
/ Pour chaque appartement, on indique s'il est loué ou non:
for (i = 0; i < appartements.length; i++) {
if (appartements[i] !== '0x0000000000000000000000000000000000000000') {
$('.panel-appart').eq(i).find('button').text('Indisponible')
    .attr('disabled', true)
    .addClass("btn-warning");
}
}
})
.catch(function(err) {
console.log(err.message);
});
```

Si c'est le cas, le bouton correspondant est désactivé, son texte changé pour « Indisponible » et sa couleur est changée. Les erreurs sont envoyées à la console.

Gestion de la fonction handleLocation() :

```
handleLocation function(event) {

event.preventDefault();
var apartId = parseInt($(event.target).data('id'));

var locationInstance;
// web3.eth.getAccounts sert à récupérer les comptes de l'utilisateur

web3.eth.getAccounts(function(error, accounts) {
if (error) {
console.log(error);
}
```

```
tfondrat@tfondrat-SATELLITE-L755:~/Bureau/thierry/prj/dapp/agenceLoc/agenceLoc$ npm run dev

> agence-location@1.0.0 dev /home/tfondrat/Bureau/thierry/prj/dapp/agenceLoc/agenceLoc
> lite-server

** browser-sync config **
{ injectChanges: false,
  files: [ './**/*.{html,htm,css,js}' ],
  watchOptions: { ignored: 'node_modules' },
  server:
   { baseDir: [ './src', './build/contracts' ],
     middleware: [ [Function], [Function] ] } }
[Browsersync] Access URLs:
 ------------------------------------
       Local: http://localhost:3000
    External: http://192.168.1.13:3000
 ------------------------------------
          UI: http://localhost:3001
 UI External: http://localhost:3001
 ------------------------------------
[Browsersync] Serving files from: ./src
[Browsersync] Serving files from: ./build/contracts
[Browsersync] Watching files...
[Browsersync] Reloading Browsers...
```

Figure 12

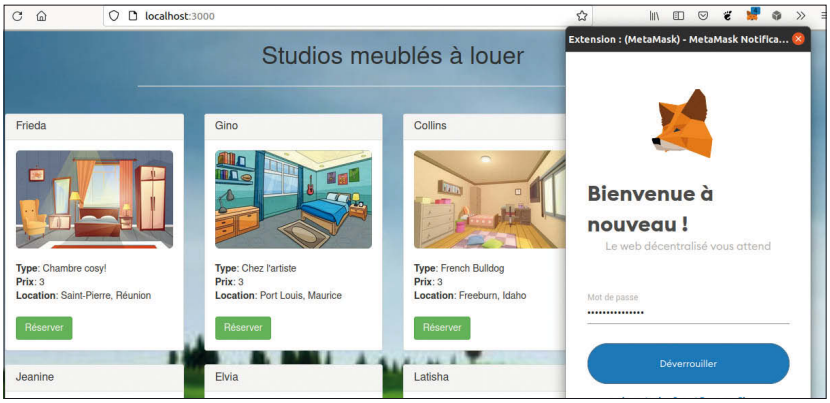


Figure 13

```
// On prend le premier compte:
var account = accounts[0];
App.contracts.Location.deployed()
    .then(function(instance) {
    locationInstance = instance;
    // Appel à la fonction loue, en transmettant le compte associé:
    return adoptionInstance.loue(apartId, {from: account});
    })
    .then(function(result) {
    return App.markLoue();
    })
    .catch(function(err) {
    console.log(err.message);
    });
});
}
};
```

Dans la callback après une vérification d'erreur, on sélectionne ensuite le premier compte.

A partir de là, on récupère le contrat déployé comme précédemment et on stocke l'instance dans adoptionInstance. Cette fois cependant, on va envoyer une **transaction** à la place d'un appel.

Les transactions nécessitent une adresse "from" et ont un coût associé. Ce coût, payé en ether, est appelé **gas**. Le coût du gas est la commission pour réaliser le calcul et/ou stocker des données dans un smart contract.

On envoie la transaction en exécutant la fonction loue() avec l'ID et un objet contenant l'adresse stockée précédemment dans account.

Le résultat de l'envoi d'une transaction est l'objet transaction. S'il n'y a pas d'erreurs, on effectue l'appel à markLoue() pour synchroniser l'UI avec notre donnée nouvellement stockée.

7) Interaction avec la dapp dans un navigateur

Le moyen le plus simple pour interagir avec notre application dapp dans un navigateur est d'installer MetaMask, une extension de navigateur..

Si vous utilisez déjà MetaMask, vous devrez changer de compte. Avant de faire cela, assurez-vous que votre phrase secrète personnelle est sauvegardée, vous en aurez besoin pour vous reconnecter à votre compte personnel (Cette phrase secrète devrait certainement rester secrète et vous ne pourrez pas accéder à votre compte sans elle ! Pour plus d'informations, vérifiez sur https://metamask.zendesk.com/hc/fr-us/articles/360015489591-Basic-Safety-and-Security-Tips-for-MetaMask)

Après avoir sauvegardé votre phrase secrète en toute sécurité, vous pouvez passer au compte test de Ganache. Pour ce faire, vous pouvez cliquer sur le logo de votre compte, puis sur "Verrouiller". Après cela, vous pouvez importer en utilisant la phrase secrète de recouvrement pour insérer le mnémonique Ganache.

1. Installer MetaMask dans votre navigateur.
2. Une fois ceci installé, un onglet s'affiche
3. Après **Getting Started**, cliquer sur **Import Wallet**.
4. Dans la boîte **Wallet Seed**, entrer le mnémonique affiché dans Ganache.

***Avertissement* :** *Ne pas utiliser ce mnémonique sur le réseau principal Ethereum. Si vous envoyez des eth à n'importe quel compte généré à partir de ce mnémonique, vous perdrez tout !*
Entrez un mot de passe et cliquez sur OK.

5. Il reste à connecter MetaMask à la blockchain créée par Ganache, cliquer sur le menu "Main Network" et choisir **Custom RPC**.
6. Dans la boite "New Network" entrer http://127.0.0.1:7545, et dans "Chain ID" entrer 1337 (ID par défaut de Ganache) et cliquer sur **Save**.

Le nom du réseau en haut va indiquer http://127.0.0.1:7545.

7. Retourner à la page Accounts.

Chaque compte créé par Ganache a par défaut 100 ether. Le premier compte a un peu moins de gas car il a été consommé au déploiement du contrat et lors des tests.

On peut maintenant démarrer un serveur web local et utiliser la dapp. On utilise la librairie lite-server fourni avec la box truffle pet-shop pour fournir nos fichiers statiques. Éventuellement, vous pouvez modifier le fichier de configuration bs-config.json. Celui-ci indique à lite-server quels fichiers inclure dans notre répertoire de base ./src pour le site web et ./build/contracts pour les artéfacts de contrats.

Une commande de dev est aussi ajoutée dans package.json. Vous pouvez la lancer par **npm run dev** **Figure 12**

Cela démarre automatiquement un navigateur ainsi qu'une fenêtre popup demandant l'autorisation de se connecter à votre wallet MetaMask. Sans approbation explicite, vous ne pourrez pas interagir avec la dapp. Cliquer sur **Déverrouiller.**

A titre d'illustration, le site montre une présentation simple. **Figure 13**

Le développement web n'étant pas le sujet de cet article, la conception du site n'a pas été détaillée.

Vous avez maintenant les outils pour devenir un développeur dapp et commencer à travailler sur des dapps plus avancées.

.NET 6 quoi de neuf ?

En novembre 2020, Microsoft annonçait la fusion du .NET Framework et .NET Core 3.x pour créer .NET 5. Une plateforme unifiée pour les développeurs combinant dev .NET, Cloud, gaming, IoT, Web, et l'IA. Le premier objectif de cette fusion était de permettre aux développeurs (Microsoft et la communauté) de travailler et développer ensemble en construisant des produits à partir d'une seule et unique base de code. Le second étant de produire un environnement d'exécution et une infrastructure .NET uniques qui peuvent être utilisés partout en ayant des comportements d'exécution et des expériences de développeur uniformes. Après ce premier pas dans l'univers de la plateforme unifiée, Microsoft annonçait la création de .Net 6 (**Figure 1**) en novembre 2021.

GAGOBA Koffi Narcisse

Ingénieur en Technologie Web et Multimédia

Avec 4 ans d'expérience, je suis consultant en Étude et développement d'applications Fullstack Microsoft .Net et Angular chez Devoteam. Je suis en mission chez VEOLIA/VEDIF. Je suis passionné par le numérique en général, les nouvelles technologies et plus particulièrement par la conception et la mise en place de solutions logicielles couvrant toute la stack technique.

Avec .Net 6, Microsoft prévoit consolider son plan de création d'un univers unifié en capitulant sur les trois principes suivants :

- Amélioration de la productivité : .Net 6 couplé avec Visual Studio 2022 offrent un rechargement à chaud. Ils mettent à disposition de nouveaux outils git, une meilleure collaboration entre les équipes, une édition intelligente du code et de nouveaux outils de diagnostic et de tests robustes.
- Simplification du développement : Avec le C# 10, la quantité de code à écrire est considérablement réduite par rapport aux versions antérieures ce qui facilite sa prise en main plus facile. En mettant en place des fonctionnalités telles que le minimal API que nous verrons plus loin dans ce dossier, il devient plus simple et plus rapide de mettre en place des microservices.
- Amélioration de la performance : .Net 6 est le framework Web full stack le plus rapide du marché. Un avantage non négligeable pour la réduction des coûts de calcul surtout pour les applications tournant dans le Cloud.

Comment donc ces trois principes ont été mis en place ?
Question à laquelle nous répondons dans la suite de ce dossier. Mais bien avant cela, nous allons préparer notre environnement à l'exécution du code **C# 10** sous **.NET 6** dans l'éditeur **Visual Studio 2022**. Nous reviendrons sur ces trois thématiques un peu plus loin.

Prérequis

Installation .NET 6

Rappelons qu'historiquement chaque version du framework se compose de deux composants : le SDK et le moteur d'exécution. Le SDK est utilisé pour créer des applications et bibliothèques .NET tandis que le moteur d'exécution lui pour l'exécution des applications .NET. Le moteur d'exécution est toujours installé avec le SDK.

Pour obtenir le SDK, nous pouvons nous rendre à l'adresse suivante https://dotnet.microsoft.com/en-us/download/dotnet/6.0 afin de découvrir les différentes releases du framework et installer la version appropriée à notre environnement ou en nous rendant sur https://dotnet.microsoft.com/en-us/download pour obtenir la version recommandée du framework. Aucune spécificité n'est requise pendant les étapes d'installation.

Une fois l'installation terminée, nous pouvons vérifier cela en tapant dans notre invite de commande **dotnet —info**. Elle nous affichera toutes les versions de SDK installées sur notre machine (**Figure 2**). Il est important de vérifier l'existence d'une version 6.X.XXX dans la section "SDK .NET installés". Il est à noter que la liste des moteurs d'exécution installés est aussi disponible en résultat de la commande.

Installation de Visual Studio 2022

En même temps qu'avec le .Net 6, Visual Studio 2022 a été lancé. Nous pouvons nous rendre à l'adresse suivante https://visualstudio.microsoft.com/fr/downloads/ (**Figure 3**) où l'on pourra faire le choix de l'édition approprié à notre besoin et procéder au téléchargement.

Après téléchargement, nous pouvons installer en prenant soin de cocher pendant l'installation les composants dont

```
C:\Users\GAGOBA>dotnet --info
SDK .NET (reflétant tous les fichiers global.json) :
 Version:   6.0.102
 Commit:    02d5242ed7

Environnement d'exécution :
 OS Name:     Windows
 OS Version:  10.0.19044
 OS Platform: Windows
 RID:         win10-x64
 Base Path:   C:\Program Files\dotnet\sdk\6.0.102\

Host (useful for support):
  Version: 6.0.2
  Commit:  839cdfb0ec

.NET SDKs installed:
  3.1.417 [C:\Program Files\dotnet\sdk]
  5.0.405 [C:\Program Files\dotnet\sdk]
  5.0.406 [C:\Program Files\dotnet\sdk]
  6.0.102 [C:\Program Files\dotnet\sdk]
```

Figure 2

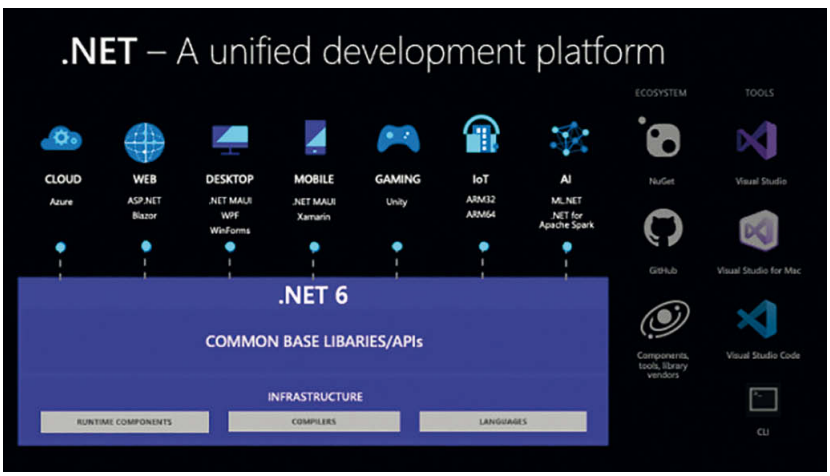


Figure 1

nous aurons besoin pour nos développements futurs (**Figure 4**). Les workloads de développement Web ASP.NET et de développement desktop .NET sont suffisants pour les besoins des programmes de ce document.

Les nouveautés

C# 10

.NET 6 embarque avec lui une nouvelle version de C#, la version 10. Avec cette nouvelle version, .Net 6 apporte de nombreuses nouvelles fonctionnalités qui aident à écrire du code plus facilement.

Global Using

On a tous déjà senti le besoin d'avoir un fichier répertoriant les espaces de noms auxquels on fait appel dans toutes nos classes ou on s'est plaint de la redondance des importations d'espaces de noms répétitifs dans nos classes. Ce ne sera plus le cas avec C#10.

Le global using nous permet de spécifier une directive using une seule fois et de l'appliquer à tous les fichiers compilés de l'application.

```
global using System;
global using System.IO;
global using System.Text;
```

Ces déclarations peuvent être placées dans n'importe quel fichier .cs.

Les espaces de noms à l'échelle fichier

Les espaces de nom à l'échelle fichier permettent de déclarer un espace de noms d'un fichier entier sans imbriquer le reste du contenu dans des accolades { … }. Vu que l'espace a une portée globale sur le fichier entier, il est à noter qu'un seul espace de nom est autorisé et sa déclaration doit être placée avant toute déclaration de Type.

```
namespace MonEspaceDeNom;

class MaClasse { ….. }
```

Les cérémonies en C#

Notre premier programme C# contenait une classe embarquée dans un espace de noms, une méthode principale et ses arguments, le tableau de chaîne de caractères string[] args. Avec le C# 10, toutes ces cérémonies deviennent facultatives. Il devient possible d'écrire directement du code dans un fichier sans se soucier des différentes cérémonies visibles sur l'image en dessous (Avant .Net 6 : **Figure 5**, après .Net 6 : **Figure 6**). Un aspect qui permettra aux nouveaux apprenants de baisser en entrée la courbe d'apprentissage.

Figure 3

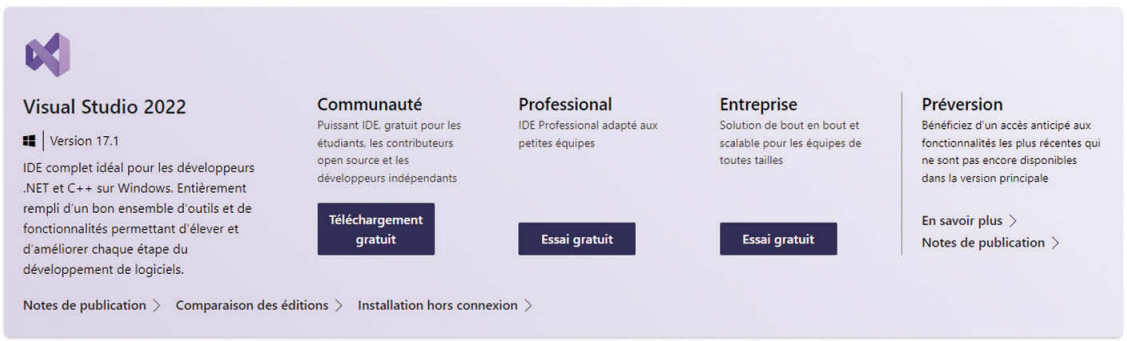
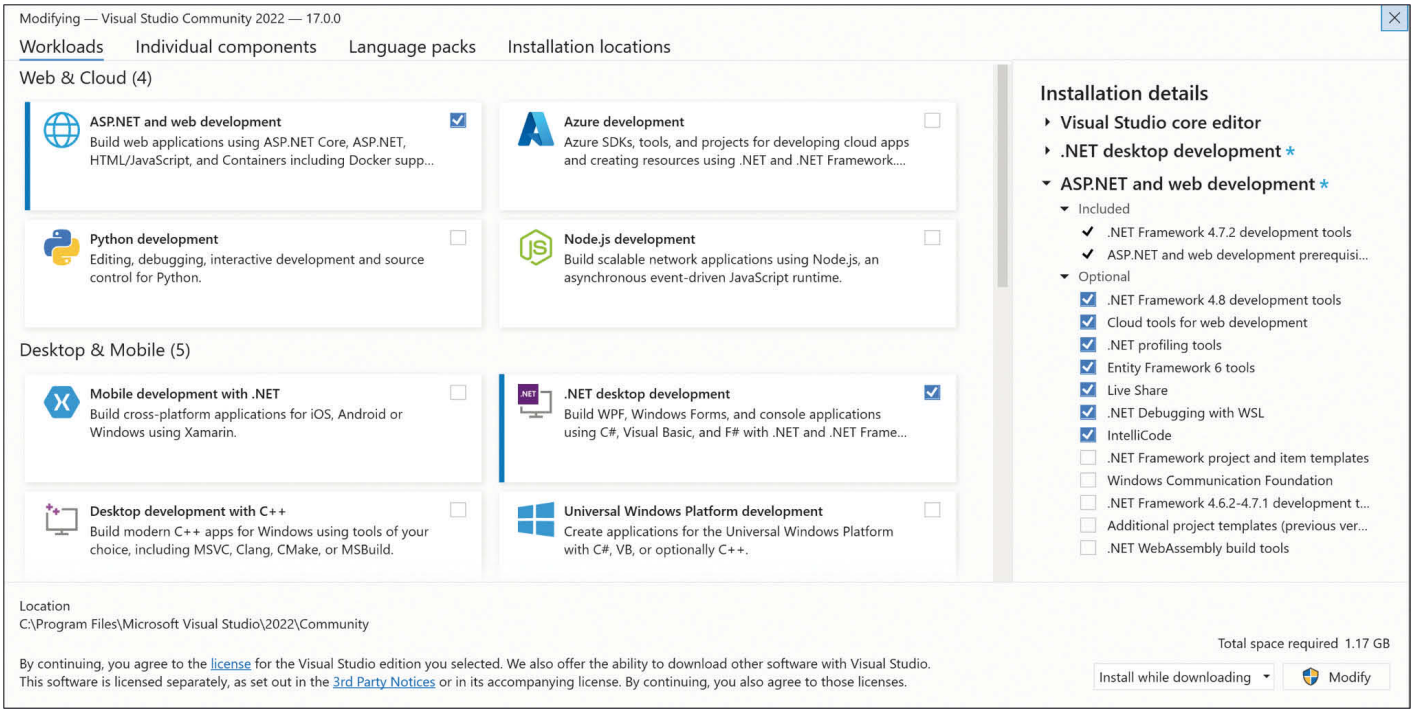


Figure 4



```
using System;

namespace Before
{
    0 références
    internal class Program
    {
        0 références
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Figure 5

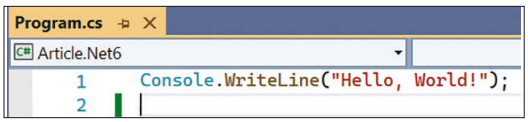


Figure 6

Constantes de chaînes de caractères interpolées

L'interpolation de chaînes de caractères est un mécanisme de formatage de chaînes de caractères. Une chaîne interpolée est une chaîne qui peut contenir des expressions d'interpolation. Lorsqu'une chaîne interpolée est résolue en une chaîne de résultats, les éléments contenant des expressions d'interpolation sont remplacés par les représentations de chaîne des résultats de l'expression.

Avant le C# 10, il était possible de concaténer des chaînes de caractères. La fonctionnalité d'interpolation de chaînes de caractères existait, mais ne permettait pas d'avoir des constantes de chaînes de caractères interpolées.

Avec le C#10, nous pouvons avoir des constantes de chaînes de caractères interpolées (**Figure 7**) et avec, naît une syntaxe plus lisible et plus pratique que le formatage composite des chaînes.

```
1 référence
public class RunClass
{
    1 référence
    public void Run()
    {
        const string platform = ".NET";
        const string language = "C#";
        const string version = "10.0";

        const string fullProductName =
            $"Platform: {platform} - Language: {language} - Version: {version}";

        Console.WriteLine(fullProductName);
    }
}
```

Record struct

Le type record est introduit en C#9. Microsoft a apporté une évolution au record en créant le type record struct avec l'arrivée du C#10. Le type record a pour but de fournir une syntaxe simple pour déclarer des objets de type référence contenant des propriétés. À la différence des classes, l'égalité dans les records ne signifie pas nécessairement l'égalité des références. Deux instances de record sont égales si les valeurs de toutes leurs propriétés sont égales. Ils peuvent être hérités à l'inverse des structures. Ils exposent également une méthode ToString() retournant chaque propriété ainsi que sa valeur, différente du ToString() d'une classe qui retourne le nom du type (**Figure 8**).

```
4 références
public record Employe(int Id, string Nom, string Prenom, int AGe);

0 références
public class Test
{
    0 références
    public static void Main()
    {
        Employe employe1 = new(1, "François", "Clément", 18);
        Employe employe2 = new(1, "François", "Clément", 18);

        Console.WriteLine(employe1 == employe2);
        // Sortie: True
        // Aurait donné False si le type Employe était une classe mais dans ce cas True
        // parce que l'égalité est testée sur la valeur des différentes proriétés

        Console.WriteLine(employe1);
        // Sortie: Person { Id = 1, Nom = François, Prenom = Clément, Age = 18 }
    }
}
```

Figure 8

Malgré que le type record soit de type référence, on peut remarquer que l'égalité entre les deux variables est vraie. Ceci s'explique par l'égalité par valeur des différentes propriétés des deux variables. Ce qui n'aurait pas été le cas s'il s'agissait du type classe.

Le type record a connu une évolution avec la création du type "record struct". Les record struct permettent de créer des records en tant que structures, et donc de type valeur. **Un record struct a toutes les fonctionnalités d'un record, mais sera alloué dans la pile comme toute structure et non dans le heap**.

Le type record struct utilise aussi donc l'égalité basée sur la valeur. Deux variables de type record struct sont égales si les définitions du type sont identiques et si, pour chaque propriété, les valeurs des deux enregistrements sont égales (**Figure 9**).

```
4 références
public record struct Employe(int Id, string Nom, string Prenom, int AGe);

0 références
public class Test
{
    0 références
    public static void Main()
    {
        Employe employe1 = new(1, "François", "Clément", 18);
        Employe employe2 = new(1, "François", "Clément", 18);

        Console.WriteLine(employe1.Equals(employe2));
        Console.WriteLine(employe1 == employe2);
        // Sortie: True
        // Un struct ordinaire n'implémente pas les opérateurs == et !=, un record struct si.
        // L'égalité est testée sur la valeur des différentes proriétés

        Console.WriteLine(employe2);
        // Sortie: Person { Id = 1, Nom = François, Prenom = Clément, Age = 18 }
    }
}
```

Extended property pattern

Avec le C#8, Microsoft a introduit les Property pattern. Un motif permettant de faire correspondre les propriétés d'un objet examiné. Le C# 10 vient améliorer ce pattern en simplifiant l'accès aux propriétés imbriquées via l'extended property pattern.

Avec le property pattern depuis C# 8 :

```
if (e is MethodCallExpression { Method: { Name: "MethodName" } })
```

Avec l'extended property pattern en C# 10, l'on obtient :

```
if (e is MethodCallExpression { Method.Name: "MethodName" })
```

La différence se situe au niveau de la correspondance des

propriétés enfants. Avec le property pattern, on observe une imbrication supplémentaire qui n'améliore pas forcément la lisibilité du code tandis qu'avec l'extended property pattern, la correspondance est faite directement via Objet.Propriété.

Mixage entre déclaration et affectation dans la déconstruction de Tuples

Un tuple offre un moyen léger de récupérer plusieurs valeurs à partir d'un appel de méthode. Cependant, une fois le tuple récupéré, nous devons faire une gestion individuelle de ces éléments. La déconstruction des tuples permet de décomposer tous les éléments d'un tuple en une seule opération. La syntaxe générale de déconstruction d'un tuple est similaire à la syntaxe qui permet d'en définir un. Les variables auxquelles chaque élément doit être affecté entre des parenthèses sont placées dans la partie gauche d'une instruction d'affectation.

Lorsque nous utilisons la déconstruction, nous sommes autorisés à assigner ou à déclarer les variables dans lesquelles nous déconstruisons. Cela signifie que ces deux déconstructions compilent :

// Attribuer des variables par déconstruction

```
int x = 0; int y = 0;
(x, y) = point;
```

// Déclarer des variables par déconstruction

```
(int x, int y) = point;
```

Avant C# 10, la déconstruction exigeait que toutes les variables soient nouvelles, ou qu'elles soient toutes déclarées au préalable.

Il n'est cependant pas possible de les mélanger dans les versions précédentes du langage. La problématique se situant dans les scénarios où l'une des variables a été préalablement déclarée dans notre code, tandis qu'une autre est obtenue par déconstruction. C# 10 lève cette contrainte et nous permet de faire les déclarations et les affectations lors de la déconstruction comme l'illustre le bout de code suivant:

```
int x = 10;
(x, int y) = point;
```

ASP .NET 6

Using implicite

Une nouvelle balise voit le jour dans le .csproj de nos projets, celle du using implicite <ImpliitUsings>.

Cette fonctionnalité force le compilateur à importer automatiquement un ensemble de using basé sur le type de projet. En prenant l'exemple d'une application console, les directives suivantes ne sont plus nécessaires à l'importation 'explicite'.

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;
```

```
<PropertyGroup>
  <TargetFramework>net6.0</TargetFramework>
  <Nullable>enable</Nullable>
  <ImplicitUsings>enable</ImplicitUsings>
</PropertyGroup>
```

Figure 10

Elle porte deux valeurs possibles (**Figure 10**), qui sont le enable et le disable.

<ImpliitUsings>enable<ImpliitUsings> pour l'activation et <ImpliitUsings>disable<ImpliitUsings> pour la désactivation.

Le framework facilite ainsi sa prise en main en prévoyant un ensemble de directives d'importation par type de projet.

Minimal API

Les API minimalistes permettent aux développeurs de créer des API à l'aide d'un seul fichier où ils peuvent configurer l'application, mais aussi écrire le code des points d'entrée ; API. La nouvelle version du framework a rendu extrêmement simple l'écriture des API REST avec le minimum de dépendances.

Dans l'exemple suivant (**Figure 11**), nous configurons l'application et créons une API d'ajout d'employés directement dans le fichier program.cs.

L'on peut vite remarquer que cette fonctionnalité mise sur la simplicité des approches et le minimalisme. En revanche, elle peut être critiquable sur certains points.

Traditionnellement la classe startup.cs permettait de configurer l'application via deux méthodes et la classe Program.cs permettait d'ajouter la configuration du serveur Web. En fusionnant ces deux fichiers de configuration et en y rajoutant les API dans un seul et même fichier, on s'expose au risque d'avoir un fichier volumineux à la longue pour des projets appelés à évoluer.

Il est important de noter que la documentation d'API est supportée pour les API minimalistes via OpenAPI.

HTTP logging

La version 6 du .NET Framework permet la configuration et l'activation de la journalisation HTTP en utilisant les méthodes d'extension ***AddHttpLogging*** et ***UseHttpLogging***. Leur utilisation active un middleware intégré permettant de capturer les informations sur les requêtes et réponses HTTP, les en-têtes et les informations sur le corps des requêtes, etc.

Pour la configurer, nous devons invoquer la méthode d'extension AddHttpLogging appartenant à l'interface

Figure 11

```
// Creation de l'app web
var builder = WebApplication.CreateBuilder(args);

// Enregistrement des services pour injection des dépendances
builder.Services.AddSingleton<IEmployeeRepo, EmployeeRepo>();
builder.Services.AddSwaggerGen();

var app = builder.Build();

app.MapPost("/api/employee", async (IEmployeeRepo repo, Employee employee) =>
{
    var employeeWithId = await repo.AddAsync(employee);
    return Results.CreatedAtRoute("GetById", new { Id = employeeWithId.Id }, employeeWithId);
});

app.MapGet("/api/employee/{id}", async (IEmployeeRepo repo, string id) => {
    var employee = await repo.GetByIdAsync(id);
    return (employee == default) ? Results.NotFound() : Results.Ok(employee);
}).WithName("GetById");

// Run the application
app.Run();
```

IServiceCollection. Le code de la **figure 12** montre comment procéder.

L'activation est faite en invoquant la méthode d'extension **UseHttpLogging** comme suit :

app.UseHttpLogging() ;

Aussi intéressante que puisse être la journalisation HTTP, elle peut nuire aux performances de l'application si l'on choisit de logger tout et n'importe quoi, notamment les corps des requêtes et réponses. Il est donc judicieux de tenir compte de l'impact sur les performances lorsque nous choisissons les champs à journaliser.

W3Clogger

Tout comme le **HttpLogging**, le **W3CLogger** est un middleware qui écrit des fichiers journaux, à la seule différence que le **W3CLogger** le fait dans le format standard du W3C. Les journaux contiennent des informations sur les requêtes HTTP et les réponses HTTP. W3CLogger fournit des journaux d'information sur les requêtes et réponses HTTP, les en-têtes, les métadonnées sur la paire demande/réponse (date/heure de début, durée), etc.

Pour l'activer, l'on utilise la méthode **UseW3CLogging** et pour configurer les éléments à logger on utilise la méthode **AddW3CLogging** (**Figure 13**).

```
services.AddHttpLogging(httpLogging =>
{
    // Quelques exemples de lignes de configuration

    // Loggé tous les champs des requêtes et réponses
    httpLogging.LoggingFields = HttpLoggingFields.All;
    // Ajouter une entête
    httpLogging.RequestHeaders.Add("Request-Header-Demo");
    // Spécifier l'encodage
    httpLogging.MediaTypeOptions.AddText("application/javascript");
    // Définir une limite de taille de la requête
    httpLogging.RequestBodyLogLimit = 4096;
    // Définir une limite de taille de la reponse
    httpLogging.ResponseBodyLogLimit = 4096;
});

app.UseHttpLogging();
```

Figure 12

```
services.AddW3CLogging(logging =>
{
    // Loggé tous les champs des requêtes et réponses
    logging.LoggingFields = W3CLoggingFields.All;
    // la taille maximale du journal en octets
    logging.FileSizeLimit = 5 * 1024 * 1024;
    // Nombre maximum de fichiers logs conservés
    logging.RetainedFileCountLimit = 2;
    // Préfixe du nom du fichier de journalisation
    logging.FileName = "MyLogFile";
    // répertoire dans lequel seront ajoutés les fichiers de journalisation
    logging.LogDirectory = @"C:\logs";
    // la période après laquelle les journaux seront vidés
    logging.FlushInterval = TimeSpan.FromSeconds(2);
});

app.UseW3CLogging();
```

Figure 13

Enrichissement du .Net BCL

La BCL représente l'une des bibliothèques standard du framework .NET. Elle fournit des types représentant les types de données CLI intégrés, l'accès de base aux fichiers, les collections, les attributs personnalisés, le formatage, les attributs de sécurité, les flux d'E/S, la manipulation des chaînes de caractères, etc. Elle contient des types de base, fondamentaux, comme System.String et System.DateTime. De nouveaux types et concepts ont été ajoutés au BCL dans cette mise à jour du framework. Les types DateOnly et TimeOnly.

Le type DateTime nous a tous contraint à gérer la partie Time dans bien de situations malgré que notre besoin se situait qu'au niveau de la date. Pour pallier cette problématique, le type DateOnly se focalise que sur les trois propriétés d'une date qui sont le jour, le mois et l'année. Ce nouveau type dispose de trois constructeurs qui sont illustrés à travers l'exemple de la **figure 14**.

Le premier initialise une date au 01-01-0001, le second au 30 Janvier 2022 avec ordre des paramètres l'année, le mois et le jour. Le troisième constructeur rajoute la notion du calendrier, dans notre exemple le calendrier grégorien.

Le type TimeOnly lui traite la partie temps, notamment de l'heure jusqu'à la milliseconde selon le besoin. Pour ce faire, il expose 4 différents constructeurs qui sont illustrés à travers l'exemple de la **figure 15**.

Le passage de ces structures vers les structures **DateTime** et **TimeSpan** sont également pris en compte à travers des méthodes telles que ToDateTime pour le DateOnly et le ToTimeSpan pour le TimeOnly.

Il est important de noter que la sérialisation de ces types n'est pas encore prise en compte dans cette version du framework. Ce que prévoit intégrer Microsoft dans la version 7. Une sérialisation de ces deux types propagerai une exception de type NotImplementedException.

```
1 référence
void DateOnlyMethod()
{
    DateOnly dateOnly = new DateOnly();
    dateOnly = new DateOnly(2022, 1, 30);
    dateOnly = new DateOnly(2021, 9, 9, new GregorianCalendar());
}
```

Figure 14

```
1 référence
void TimeOnlyMethod()
{
    TimeOnly timeOnly = new TimeOnly();     // 00:00
    timeOnly = new TimeOnly(3060000000000); // 08:30
    timeOnly = new TimeOnly(7, 25);         // 07:25
    timeOnly = new TimeOnly(8, 30, 0);      // en rajoutant les secondes
}
```

Figure 15

.NET MAUI

Rappelons qu'avec le .NET 5, Microsoft entamait le processus d'unification de la plateforme .NET, en réunissant .NET Core et Mono/Xamarin dans une seule bibliothèque de classes de base BCL et une seule chaîne d'outils SDK.

.NET MAUI acronyme de .NET Multi-platform App UI est le nom de la nouvelle solution améliorée en tant que cadre d'interface utilisateur d'applications multi plateformes pour créer des applications natives multi plateformes avec .NET pour Android, IOS, macOS et Windows (**Figure 16**).

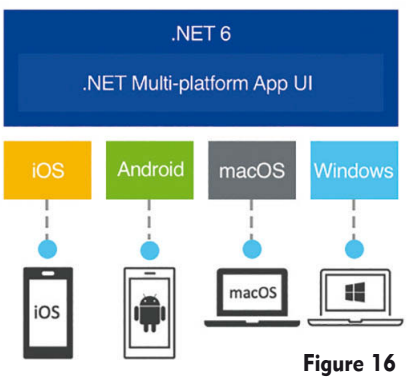


Figure 16

Malheureusement dans la release du .NET 6 de novembre 2021, .NET MAUI est le gros absent. Afin d'offrir à ses utilisateurs la meilleure expérience, la meilleure performance et la meilleure qualité, Microsoft annonçait en septembre 2021

la mise à disposition du .NET MAUI au début du deuxième trimestre de 2022, quelques mois après la publication du .NET 6. La version preview du .NET MAUI est cependant disponible pour utilisation. Microsoft ne donne aucune garantie dessus tant qu'elle pourra considérablement être modifiée avant la sortie de sa release.

Linq

De nombreuses méthodes LINQ ont été ajoutées dans .NET 6. Ces méthodes sont DistinctBy (), UnionBy (), IntersectBy (), ExceptBy (), MaxBy (), MinBy () and Chunk () que nous détaillerons :

- DistinctBy () retourne les éléments distincts d'une séquence selon une fonction de sélection de clé.
- ExceptBy () retourne la différence d'ensemble de deux séquences selon une fonction de sélection de clé.
- IntersectBy () retourne l'intersection de deux séquences selon une fonction de sélection de clé.
- UnionBy () produit l'union de deux séquences selon une fonction de sélection de clé.
- MinBy () et MaxBy () retournent la valeur minimale pour le MinBy ou maximale pour le MaxBy dans une séquence générique selon une fonction de sélection et éventuellement d'un comparateur de clés spécifiées.
- Chunk () divise les éléments d'une séquence en morceaux de taille au plus égale à la valeur en paramètre.
- Et d'autres méthodes ayant leur équivalent dans System.Linq.Queryable.

Performance

.NET 6 représente la version la plus performante de toutes les versions du framework depuis son avènement. **Stephen Toub**, développeur au sein de l'équipe .NET de Microsoft a rédigé un super article sur les performances du .NET 6 disponible sur le blog de Microsoft. L'article en question a été rédigé sur la base des 550 Pull requests traitant des sujets de performance sur les 6500 que la branche principale du projet .NET 6 comptait. Dans cette section de notre dossier, nous décrirons succinctement quelques-unes des performances les plus significatives.

Compilateur JIT

Rappelons que JIT acronyme de Just In Time est un mécanisme de compilation utilisé pour traduire le langage intermédiaire en code assembleur au moment de l'exécution. Dans .NET 6, d'importantes améliorations ont été apportées au compilateur. Trois de ces améliorations sont axées sur l'inlining, la dévirtualisation et l'optimisation guidée par le profil.
L'inlining est le remplacement d'un appel de méthode par le code de cette méthode dans la méthode appelante. Il a pour objectif majeur d'exposer le contenu de la méthode appelée au contexte de l'appelant, permettant une optimisation ultérieure en exécution.
Il n'a pas que des avantages. Par principe, elle augmente la taille du code assembleur. Son utilisation de façon démesurée impliquerait une grande consommation de mémoire. Les ordinateurs utilisent des caches d'instructions très rapides où sont stockées les codes à exécuter dans le cadre d'une réutilisation. En pratiquant l'inlining, le code de la méthode appelée n'est pas nécessairement mise en cache et peut entraîner un ralentissement d'exécution. Microsoft a pris en compte ces paramètres en optimisant la compilation JIT via une évaluation méthodique de ce qu'il faut mettre inline ou pas au travers d'une variété d'heuristiques.
L'inlining va de pair avec la dévirtualisation qui, elle, est l'acte par lequel le JIT prend un appel de méthode virtuelle ou d'interface et détermine statiquement la cible finale réelle de l'invocation en émettant un appel direct à cette cible et par là économiser le coût de la répartition virtuelle. Une fois dévirtualisée, la cible peut également subir l'inlining pour davantage d'optimisations.
L'Optimisation orientée profil de l'expression anglaise *Profile-guided optimization (PGO)* a déjà été mise en œuvre dans une variété de piles de développement et a existé dans .NET sous de multiples formes au fil des années, notamment le PGO statique. L'amélioration majeure apportée en .NET 6 est l'optimisation orientée profil dynamique.
Depuis .NET Core 3, le principe de la compilation par étage est disponible en .NET. La compilation commence au niveau 0 ou le JIT applique très peu d'optimisations via une compilation assez rapide. Le code résultant de cette première étape de compilation comprend des données de suivi pour compter la fréquence d'appel des méthodes. Le JIT met par la suite en file d'attente les éléments de sortie du niveau 0 dépassant un certain seuil pour une compilation de niveau 1. À cette étape, toutes les optimisations possibles sont faites par le JIT en tirant des leçons de la compilation de niveau 0. À titre d'exemple une variable statique en lecture seule peut devenir une constante du fait que sa valeur aura déjà été calculée au moment où le code du niveau 1 est compilé. Avec le PGO dynamique, le JIT peut maintenant effectuer une instrumentation plus poussée pendant le niveau 0, pour suivre non seulement le nombre d'appels, mais aussi toutes les données intéressantes qu'il peut utiliser pour l'optimisation guidée par le profil et les utiliser pendant la compilation du niveau 1. Par défaut dans .NET 6, le PGO dynamique est désactivé. Son activation est faite en définissant la variable d'environnement **DOTNET_TieredPGO** à 1.

Les Entrées et Sorties

Les améliorations apportées aux Entrées et Sorties de fichiers se traduisent par une réduction du temps d'exécution, de la latence et de l'utilisation de la mémoire. Pour ce faire, nous allons nous référer à l'étude et comparaison faite par Microsoft à travers l'utilisation de quelques méthodes clés dans le but d'observer le temps d'exécution et la mémoire utilisée lors d'opérations d'entrées sorties sur le framework .NET 5 et sur le .NET 6. Notons que les paramètres temps d'exécution et utilisation de la mémoire dépendent de la quantité de données traitées tel dans le comparatif suivant que dans ceux d'après.

Méthode	Temps d'exécution		Mémoire	
	.NET 5	.NET 6	.NET 5	.NET 6
ReadAsync	25,101 ms	13,108 ms	1 094 096 B	382 B
Read	219,388 ms	83,070 ms	3 062 741 B	2 109 867 B
WriteAllTextAsync	1,590 ms	1,143 ms	23 KB	15 KB
Compress	1 050,2 us	786,6 us	-	-

Reflection

Des améliorations significatives sont également apportées aux méthodes réflexives comme le montre le comparatif suivant.

Méthode	Temps d'exécution		Mémoire	
	.NET 5	.NET 6	.NET 5	.NET 6
GetCustom AttributesData	73,58 ns	69,59 ns	48 B	-
IsDefined	292,01 ns	252,00 ns	48 B	-
GetCustomAttributes	2 406,51 ns	446,29 ns	1 056 B	128 B
Invoke	141 ns	123,1 ns	104 B	64 B
Create	26,350 ns	9,439 ns	24 B	24 B

Collections et LINQ

Les collections et le composant LINQ ont été rendus de plus en plus performants au fil des versions du framework. La version 6 apporte des améliorations, certaines sous forme d'optimisation d'API et d'autres sous forme de nouvelles API, ces dernières abordées un peu plus haut dans ce document.

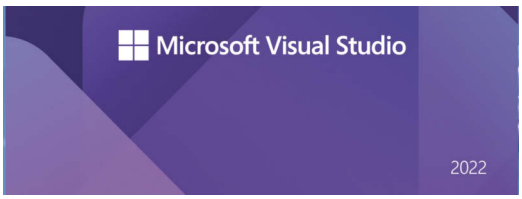
Méthode	Temps d'exécution		Mémoire	
	.NET 5	.NET 6	.NET 5	.NET 6
Min	5,207 ms	4,291 ms	-	-
Max	4,926 ms	4,222 ms	-	-
Count	12,866 us	5,778 us	32 B	32 B
SequenceEqual	5 421,1 us	150,2 us	-	-
Get Collection Item	16,296 ns	9,457 ns	-	-
Distinct on collection and Count	5,154 ms	2,626 ms	5 MB	2 MB

Ces tableaux comparatifs ne sont sans doute pas assez significatifs pour tirer de grandes conclusions, mais présentent tout de même des chiffres intéressants sur les améliorations de performances qu'on peut obtenir en migrant vers le .NET 6.

Les Outils

Cette section n'est pas spécifique à la version du langage, mais nécessite un petit focus.

En même temps que la version 6 du framework .Net, Microsoft a sorti une nouvelle version de Visual Studio 2022 (**Figure 17**).

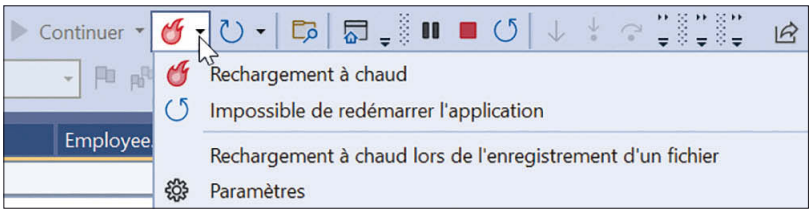


Visual Studio 2022 est une version purement 64 bits, un choix qui d'après les responsables du produit permettrait de tirer profit au mieux de toutes les ressources systèmes. Il embarque un certain nombre d'améliorations en matière d'édition et de débogage. À titre d'exemple, il offre la fonction Hot Reload, qui permet aux développeurs de modifier leur code source pendant que leurs applications s'exécutent dans l'éditeur. Il dispose également de capacités de Live Preview et de tests multi plateformes sur Linux, parmi d'autres fonctionnalités nouvelles et améliorées.

Le rechargement à chaud (Hot Reload)

Il s'agit incontestablement de l'une des fonctionnalités phares publiées dans .NET. La petite modification ou la petite faute de frappe qui impliquait tout un redémarrage de notre application, choses assez frustrantes qu'on a tous certainement connu. Avec le Hot Reload nous pouvons modifier du code en cours d'exécution et apercevoir directement les modifications sans avoir à redémarrer notre application ou mettre manuellement en pause ou d'atteindre un point d'arrêt, et ce quelque soit le type d'application. La fonctionnalité existait certes dans Visual Studio 2019, mais présentait quelques lacunes en termes de performance. Celle embarquée avec Visual Studio 2022 est beaucoup plus performante. Il existe évidemment quelques limitations, notamment si l'on modifie le démarrage d'une application ou tout code en exécution unique. Dans ce cas, nous devons redémarrer notre application pour que les modifications soient prises en charge.

En exécutant notre application, dans la barre des tâches nous pouvons apercevoir l'icône en forme de flamme (c'est vraiment du Hot), c'est le Hot Reload ou Rechargement à chaud (**Figure 18**).



Le Hot reload est possible directement depuis la ligne de commande en exécutant la commande : dotnet watch.

Web Live Preview

Le Web Live Preview permet une correspondance en temps réel entre notre code source et le rendu HTML de nos applications Web ASP.NET Framework. En d'autres termes, elle nous permet de faire des modifications directement depuis le rendu et d'observer les modifications dans le code correspondant et beaucoup plus. Pour tirer parti de cette fonctionnalité avant Visual Studio 2022, il fallait installer une extension Visual Studio portant le même nom que la fonctionnalité. Depuis Visual Studio 2022, elle est par défaut embarquée avec l'éditeur et des améliorations sont apportées afin de supporter également les applications WPF via le XAML Live Preview.

Pour l'activer ou le désactiver, il faudra se rendre dans **Outils -> Options -> Web Live Preview (préversion)** et mettre à true ou false les options désirées. Il faudra par la suite cliquer sur Ok et redémarrer Visual Studio pour tirer profit du Web Live Preview.

Le remote testing

Plein de nouveautés ont été ajoutées à l'éditeur pour améliorer l'expérience des tests. La possibilité que l'éditeur émette un son à la fin d'exécution des tests, celle de naviguer entre un fichier de test et l'explorateur de test via le menu contextuel disponible au clic droit dans le fichier ou depuis l'explorateur de solution et celle de faire un Console.WriteLine() dans une méthode de test; la sortie est affichée dans le panel de détail des tests.

La fonctionnalité la plus intéressante est celle du remote testing. Elle permet de connecter l'éditeur à des environnements

distants pour exécuter et déboguer des tests. Elle est très utile pour les développeurs multi-plates-formes car, avec elle, nous pouvons exécuter des tests Linux directement à partir de Visual Studio en connectant l'explorateur de tests à l'environnement distant. Il peut s'agir de conteneurs Linux, de WSL, et même de connexion SSH.

III. Impacts sur les technologies non prises en charge

Avec cette nouvelle version du framework, des impacts significatifs sont à signaler lors d'une mise à niveau de certains types d'applications. Pour les applications utilisant les formulaires Web ou Windows Workflow Foundation, l'application devra être réécrite en .Net 6.

ASP.NET Web forms

Depuis .NET 5, la nouvelle orientation de Microsoft est axée sur les bases du .NET Core, et non sur celles du .NET Framework, dont les formulaires Web font partie. Il est donc impossible d'exécuter nos applications ASP.NET Web Forms en utilisant le runtime .NET Core. Mais cela ne signifie pas que Microsoft s'est "débarrassé des formulaires Web".

ASP.NET Web Forms n'est clairement plus une option pour les nouveaux développements. Il est certes à éviter, mais pas mort. Nos applications en Web Forms continueront de fonctionner tant que nous aurons du personnel pour les supporter, mais tout comme les applications ASP classiques, elles deviendront de plus en plus fragiles à mesure qu'elles vieillissent. En bref, nous pouvons continuer à exécuter des applications Web Forms, mais faire le choix de nouvelles technologies pour de nouveaux développements. Pour ce faire, il est recommandé de commencer à tester **Razor Pages**, **Blazor** et **ASP.NET Core MVC** et voir quelle technologie comblera au mieux les lacunes.

Windows workflow foundation

Windows Workflow Foundation nécessitera un examen beaucoup plus long. Non seulement il n'est pas porté, mais les composants (comme WCF) utilisés en conjonction n'ont pas été portés également. CoreWF (le remplacement recommandé de .NET 5) peut devenir assez bon pour de nombreux utilisateurs, mais il n'est pas prêt pour la production dès maintenant.

WCF

Les applications WCF devront également être réécrites. Des outils facilitent cette réécriture, l'un d'entre eux est une extension de Visual Studio au nom de **Visual ReCode**. Il aide à la réécriture des applications utilisant le WCF vers le gRPC pour .NET Core et .NET 5+ dès maintenant. Le gRPC est un cadre RPC (Remote Procedure Call) moderne, open source et performant fonctionnant dans n'importe quel environnement.

.Net 6 et Azure

Les principaux outils PaaS (platform-as-a-service) Azure ont très rapidement adopté .NET 6. Les principaux services affectés sont Azure Functions, Azure App Service et Azure Static Web Apps. La prise en main de ces services ne devrait pas être trop difficile, tant la compatibilité est ascendante entre les versions du framework. Pour adopter la nouvelle version du framework, il aura lieu de faire une mise à niveau des environnements de développement et de test locaux avant une mise à niveau de l'application Azure.

Conclusion

.Net 6 révolutionne considérablement le développement d'applications. Un point clé à noter est son support à long terme étalé sur 3 ans. Un schéma qui se poursuivra pour chaque version paire de .NET (6, 8, 10, etc.). Pour de nombreuses organisations, cela fait de .NET 6 la première version fiable du nouveau .NET qu'elles sont prêtes à utiliser, car elles s'en tiennent strictement aux versions LTS (Support Long Terme). La sortie de .NET 6 constitue donc un changement de paradigme assez important.

Notons que dans ce dossier, plein d'autres nouveautés n'auront pas été traitées. Il en existe énormément qui auraient mérité le détour; telles que les améliorations apportées à la classe Process, l'amélioration du type struct, la génération sécurisée des nombres aléatoires, le support du protocole HTTP/3 qui vise à apporter des avantages en termes de performances, principalement pour les utilisateurs mobiles ou les connexions peu fiables ou encore les améliorations de performance autour des entrées/sorties de fichiers. Elles auraient toutes mérité le détour, mais le but n'étant pas de faire une liste exhaustive des nouveautés, mais plutôt de montrer la belle facette révolutionnaire que cette version du framework apporte à nos développements.

Le 17 février dernier, Microsoft en célébrant sa communauté et ses 20 ans d'innovation annonçait la prochaine étape du framework .NET; .NET 7 via sa version preview 1. .NET 7 s'appuie sur les fondations établies par .NET 6, qui sont : un ensemble unifié de bibliothèques de base, de runtime et de SDK, une expérience de développement simplifiée et une productivité accrue des développeurs. Les principaux domaines d'intérêt de .NET 7 comprennent une meilleure prise en charge des scénarios cloud native, des outils facilitant la mise à niveau des projets existants et la simplification de l'expérience du développeur en facilitant le travail avec les conteneurs.

Ceci dit, on en déduit assez clairement que .NET 6 ne représente que la fondation des futures évolutions du framework; et c'est tout à notre honneur.

Ressources et liens

https://docs.microsoft.com/en-us/dotnet/core/whats-new/dotnet-6
https://dotnet.microsoft.com/en-us/download
Install Visual Studio | Microsoft Docs
https://www.youtube.com/watch?v=E4frjpRqLLM
https://betterprogramming.pub/upcoming-features-in-c-10-f4471203dacc
https://anthonygiretti.com/2021/07/27/introducing-c-10extended-property-patterns/
https://www.iqlance.com/whats-new-in-microsoft-net-6/
Microsoft ships .NET 6 - Kiratas
What's new in .NET 6 | Microsoft Docs
https://visualrecode.com/blog/net6-look-ahead/
https://blog.inedo.com/dotnet/net5-web-forms
Update on .NET Multi-platform App UI (.NET MAUI) - .NET Blog
Hot Reload In C# .NET 6 / Visual Studio 2022 - .NET Core Tutorials
Announcing .NET 7 Preview 1 - .NET Blog
Performance Improvements in .NET 6 - .NET Blog

Xamarin.Forms est mort, vive .NET MAUI !

De nos jours, la problématique concernant les applications supportant plusieurs plates-formes ou systèmes d'exploitation est souvent au cœur des discussions des éditeurs, mais aussi parfois des utilisateurs qui veulent retrouver les mêmes interfaces et la même ergonomie quel que soit leur appareil. Concernant les éditeurs, certains font le choix de dupliquer leur code pour produire des applications natives pour chaque plate-forme, et d'autres décident de mutualiser leur code source pour n'écrire leur application qu'une seule fois pour plusieurs plates-formes.

Nous ne détaillerons pas ici les avantages et les inconvénients de l'une ou l'autre des décisions puisque nous allons nous focaliser sur la réponse de Microsoft, son nouveau framework .Net MAUI, successeur de Xamarin. Nous détaillerons comment cette solution s'intègre dans l'écosystème .NET et la comparerons à ses concurrents actifs sur le marché.

Présentation générale

.Net MAUI ! Ce nom exotique est en fait simplement l'acronyme de ".Net Multi-platform Application UI". C'est donc un composant logiciel qui fournit une interface utilisateur pour différents systèmes d'exploitation. Comme son grand frère Xamarin le faisait avant lui, il prend en charge les systèmes d'exploitation mobiles que sont Android et iOS, mais en plus, et c'est une nouveauté, il prend en charge les systèmes d'exploitation Desktop Windows et macOS. Il permet ainsi de développer des applications natives quelle que soit la plate-forme. De plus, comme nous le verrons plus tard, il existe aussi des synergies avec Blazor pour créer une application hybride multi-plate-forme native et Web.

Tout ceci permet donc d'avoir à partir d'un même code source une distribution sur la grande majorité des supports aujourd'hui utilisés sur le marché. Mais est-ce vraiment une nouveauté ? En réalité, c'est en fait une évolution du projet Xamarin.Forms qui permettait déjà de produire des applications pour Android et iOS.

.Net MAUI a connu sa première pré-version en mai 2020. Le projet n'est pas encore disponible de manière générale puisque pour y accéder il faut pour le moment utiliser Visual Studio 2022 dans sa version "Preview". Tout récemment, le 12 avril, la version Release Candidate a été publiée. Une disponibilité générale est prévue pour la fin du 2e trimestre 2022.

Actuellement .Net MAUI prend en charge les plates-formes suivantes (**Figure 1**) :

- Windows Desktop ainsi que la plate-forme UWP (Universal Windows Platform) grâce à WinUI 3 (Windows UI Library)
- Android à partir de sa version 5.0 (API 21)
- iOS à partir de sa version 10
- macOS à partir de sa version 10.13 (via Mac Catalyst)

Microsoft annonce apporter une attention particulière pour conserver des features équivalentes sur toutes les plates-formes.

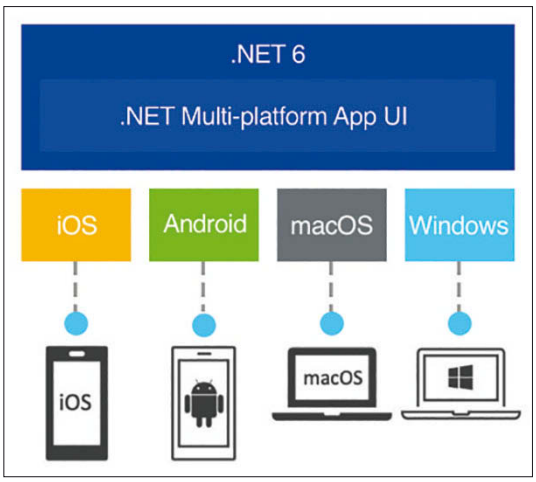


Figure 1 : *Les plates-formes supportées par .NET MAUI*

A ces plates-formes s'ajoutent aussi :

- Tizen, le système d'exploitation maison de Samsung dont le support est directement implémenté par la firme coréenne
- Linux de manière non officielle (support effectué par la communauté)

Une expérience unifiée par rapport à Xamarin.Forms

.NET MAUI profite de l'unification des différents runtimes permises par .NET 6 et offre ainsi une API de haut niveau unifiée qui adresse les différentes plates-formes (Android, iOS, Windows, macOS). Pour cela, il continue de s'appuyer sur l'ensemble d'API fourni par Xamarin Essentials, renommé pour l'occasion en .NET MAUI Essentials. Ces API permettent d'accéder aux fonctionnalités des appareils natifs : connectivité, accès aux capteurs comme l'accéléromètre ou la lampe de poche, envoi de SMS/mails… Il reste cependant possible de travailler avec les APIs natives si nécessaire.

La grosse nouveauté se situe dans le fait qu'avec .Net MAUI, il n'y a qu'un seul projet qui pointe sur différentes plates-formes quand, pour Xamarin.Forms, il y avait un projet par plate-forme. En Xamarin.Forms, il était possible d'extraire des librairies afin de mutualiser une grande partie du code et de les référencer dans les différents projets. Cependant

Gaëtan VIOLA-NGUYEN

Forgé par 5 ans d'alternance en PME dans un environnement Microsoft, via un DUT puis un diplôme d'ingénieur, je suis aujourd'hui développeur .Net. A 25 ans, marié, papa de 2 garçons, mais surtout avide de nouvelles expériences, je jongle avec le temps libre qu'il me reste pour assouvir ma curiosité dans le monde de l'informatique dans sa globalité.

Christophe HERAL

-@ChrisHeral

Développeur sur les technologies Microsoft depuis 2003 et plongé dans l'agilité depuis 2009, je m'intéresse plus particulièrement aux pratiques techniques (Software Craftsmanship, DevOps) et à la transformation produit. Je partage mon expérience et mes convictions en travaillant au cœur des équipes de développement.

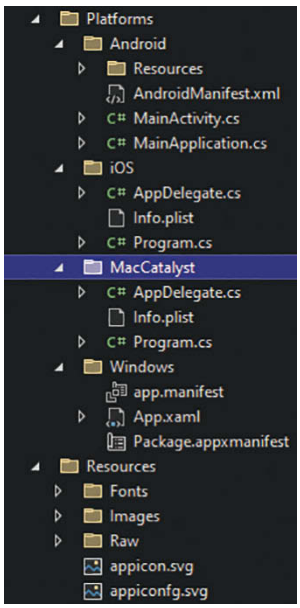


Figure 2 : *le Single Project*

certains éléments restaient dupliqués sur chacun des projets, c'était notamment le cas des différentes polices d'écritures ou même des images. Tout ceci est révolu avec .Net MAUI (**Figure 2**).

A propos des images, la prise en charge du format SVG sera intégrée de manière native dans .Net MAUI, ce qui simplifiera largement son utilisation et permettra d'avoir un rendu visuel totalement basé sur une approche vectorielle.

Pas de panique pour autant si vous migrez depuis Xamarin.Forms, il vous est possible de continuer à travailler avec vos différents projets sous MAUI.

Rechargement à chaud

Tout comme .Net 6, MAUI intègre le rechargement à chaud (hot reload). Le confort et la facilité de développement s'en trouvent accrus. Les changements opérés dans un fichier XAML sont immédiatement reportés dans l'application en cours d'exécution, même sans sauvegarde.

Concernant le code écrit en C#, il suffit de cliquer sur le bouton Hot Reload de Visual Studio 2022 pour que les changements soient effectifs tout en conservant bien sûr l'état actuel de l'application.

Intégration avancée dans Visual Studio 2022

Microsoft a enrichi l'intégration de .NET MAUI avec Visual Studio 2022 afin d'améliorer l'expérience du développeur. Par exemple, une fenêtre miroir de l'application lancée en debug vient s'intégrer directement dans Visual Studio, ce qui est pratique pour les développeurs qui travaillent sur un seul écran. Il s'agit véritablement d'un miroir, ce qui signifie qu'on l'on peut interagir avec et que l'état est le même que celui de l'application lancée en debug. Il est également possible d'utiliser des outils de Visual Studio comme les règles qui vont nous permettre de mesurer différentes tailles en pixels directement sur une application en cours d'utilisation (**Figure 3**).

De nouveaux patterns de développement

Xamarin utilisait déjà le pattern MVVM (Model View View-Model) avec l'utilisation du XAML depuis sa sortie en 2006. Avec .Net MAUI, Microsoft introduit la prise en charge de patterns plus récents comme MVU et Blazor. MVU (Model-View-Update) rompt avec le pattern MVVM en rendant l'état de l'application immuable, en apportant de la centralisation et de l'explicitation (l'état ne peut être initialisé et mis à jour qu'à un seul endroit). MVU empêche aussi les problèmes d'accès concurrents. Le XAML est toujours pris en charge, le projet par défaut est d'ailleurs toujours généré avec du XAML. Du point de vue du code, MVU ressemble beaucoup à la sémantique du Dart pour Flutter. La **figure 4** montre un code en version XAML respectant le pattern MVVM, celui de la **figure 5** montre une version en C# respectant le pattern MVU. Ces 2 bouts de code produisent exactement le même rendu graphique.

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="MauiAppArticle.NewPage2"
             Title="NewPage2"
             BackgroundColor="White">
    <StackLayout>
        <Label Text="Welcome to .NET MAUI!"
               VerticalOptions="Center"
               HorizontalOptions="Center" />
    </StackLayout>
</ContentPage>
```

Figure 4 - *MVVM*

```
namespace MauiAppArticle;

public class NewPage1 : ContentPage
{
    public NewPage1()
    {
        Content = new StackLayout
        {
            Children = {
                new Label {
                    HorizontalOptions = LayoutOptions.Center,
                    VerticalOptions = LayoutOptions.Center,
                    Text = "Welcome to .NET MAUI!"
                }
            }
        };
    }
}
```

Figure 5 - *MVU*

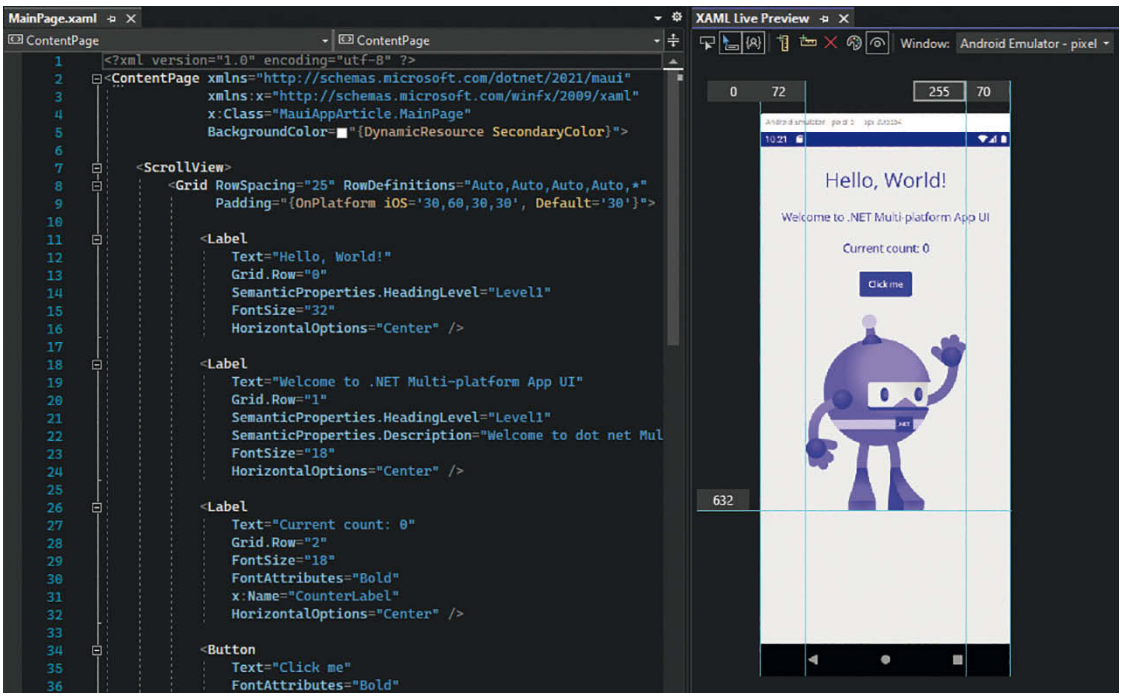


Figure 3 - *XAML Live Preview*

C'est alors une affaire de goût : MVU apporte l'immutabilité, irrésistiblement liée à l'univers des langages fonctionnels comme Elm dont il est issu. Cela reste cependant une option et les développeurs habitués à MVVM peuvent continuer à utiliser ce pattern.

Blazor a été pensé comme une alternative à JavaScript pour les développeurs de l'écosystème C# et .Net en se basant sur la technologie WebAssembly. La possibilité d'utiliser Blazor avec .Net MAUI suit la logique qui avait fait le succès d'Electron JS, c'est-à-dire utiliser une méthode de développement respectant les standards du Web pour développer des applications Desktop. Grâce à BlazorWebView, .Net MAUI ajoute à cela les plates-formes mobiles. Il est donc possible de développer une application Android avec du code HTML.
De plus, il est possible d'intégrer une application Blazor existante pour l'intégrer à un projet .Net MAUI très facilement.

Découplage de l'interface utilisateur et du système d'exploitation

Un des principaux apports de .Net MAUI est aussi le découplage qui a été opéré par les équipes de Microsoft au niveau de l'architecture des moteurs de rendu. En effet, il est maintenant question de "slim renderers". Pour comprendre la différence, il faut voir comment fonctionnait Xamarin.Forms auparavant. Les moteurs de rendu qui produisaient les vues natives dépendaient directement des contrôles de Xamarin.Forms. Avec MAUI, une interface a été introduite entre les contrôles MAUI et les moteurs de rendu natifs.
Si l'on prend l'exemple du bouton sur la **figure 6**, il faut considérer que l'interface IButton n'était pas présente dans Xamarin.Forms.

Ce changement permet de dissocier la responsabilité du rendu spécifique à la plate-forme, qui est géré par chaque moteur de rendu, et l'implémentation de l'interface utilisateur.
Par conséquent, cela permet de créer des implémentations alternatives de l'interface utilisateur, mais qui restent basées sur .Net MAUI pour la gestion multi-plate-forme.

Le projet Comet

Ainsi, il y a déjà un projet expérimental qui profite de ce changement, Comet, et qui met à disposition un pattern MVU.
Au lieu de se baser sur XAML, Comet permet de définir entièrement ses interfaces graphiques avec du code C# Fluent. Les API Fluents, populaires depuis quelques années, se basent sur des méthodes d'extension pour permettre d'appeler les méthodes de manière chaînée. Cela signifie qu'on pourra ajouter un élément graphique de type texte argenté en taille de police de 10 et avec une marge interne de 30 avec la syntaxe suivante :

```
new Text("Hello World!").FontSize(10).Color(Color.Silver).Padding(30);
```

Comet tire ainsi complètement profit du hot reload : pendant

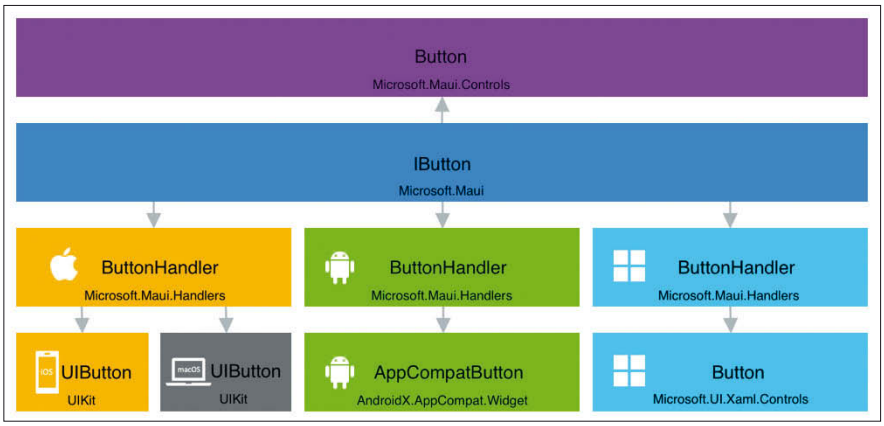


Figure 6 - *Découplage des contrôles de la plate-forme*

que l'on écrit ses modifications d'interface dans le code C#, l'affichage visuel se met à jour à chaque sauvegarde de code dans la fenêtre graphique.

Composants visuels

MAUI permet nativement de choisir un thème parmi les différents visuels courants (Cupertino, Material Design, Fluent IO) et offre également la possibilité d'étendre les contrôles de manière plus facile que Xamarin Forms (notamment sans utiliser de custom renderer). Dès sa sortie officielle, il est fort probable que certains éditeurs de composants profitent de cette possibilité pour proposer leurs propres visuels compatibles MAUI.

Intégration dans l'écosystème Microsoft

Microsoft met tout en place avec MAUI pour que les développeurs habitués aux technologies Microsoft puissent faire du développement mobile multi-plates-formes dans leur écosystème préféré tout en capitalisant sur leurs connaissances.
L'important pour eux était d'intégrer complètement Xamarin dans l'univers .NET à partir de .NET 6 et de proposer la même expérience pour tous les développeurs : que cela soit via Visual Studio, via VS Code (où le debug est également supporté) ou via la CLI.
.NET 6 étant par ailleurs une version LTS (Long Term Support - donc supportée jusqu'à fin 2024), cela devrait rassurer les entreprises envisageant de migrer sur cette technologie.

Pour les développeurs Xamarin.Forms, la migration sera facile. Pour les nouveaux, la marche sera un peu plus importante, mais ils ne seront pas perdus, car MAUI continue de s'appuyer sur le couple XAML/C# qui survit à l'enchaînement des technologies depuis l'apparition de WPF en 2006.

Profiter de toutes les features des dernières versions de C# est très agréable pour des développeurs Back ou habitués aux frameworks Web Microsoft. Pour autant, les développeurs préférant la programmation fonctionnelle et aficionados de F# seront ravis d'apprendre que .Net MAUI supporte également ce langage.

.NET MAUI vs ses concurrents

Si MAUI a copié le rechargement à chaud sur ses concurrents, c'est que c'était une feature incontournable dans l'écosystème de développement mobile actuel et l'un des points noirs de Xamarin dans les comparaisons passées avec ses concurrents.

Si MAUI propose une syntaxe en C# Fluent pour définir ses interfaces, c'est qu'il souhaite attirer tous les développeurs qui ont trouvé que c'était une super feature de Swift UI ou Flutter.

Si MAUI souhaite développer une communauté Front autour d'un langage bien connu des développeurs, c'est parce qu'il pense que la réussite de React Native a été de se baser sur une forte communauté JavaScript, alors que le ticket d'entrée pour Flutter est plus important avec la nécessité de monter en compétence sur Dart.

Si Microsoft nous propose un produit mature, c'est que ce n'est pas vraiment un nouveau produit, mais une grosse évolution de Xamarin.Forms agrémenté de tous les avantages qui lui manquaient par rapport à ses concurrents.

Si Microsoft investit pour que ses concurrents comme Flutter puissent aussi construire des applications Windows, c'est que la société ne met pas tous ses œufs dans le même panier, refroidie par l'expérience du passé (morts successives de Silverlight, WinRT, Windows Phone,…).

Il reste cependant une grosse spécificité dans l'approche de ce marché par Microsoft : au lieu d'avoir un outil phare qui cible exclusivement les développeurs Front, la firme de Redmond a une autre stratégie : capitaliser sur son langage de prédilection très répandu.

Avec .Net MAUI, C# ajoute finalement une nouvelle corde - le développement d'applications natives - à son arc, qui en présentait déjà de nombreuses autres : le développement d'applications Web, d'API REST ou même de jeux vidéo avec Unity.

Conclusion

L'ambition affichée par Microsoft est clairement de fournir des outils modernes aux programmeurs de sa communauté existante pour qu'ils n'aient pas à aller voir ailleurs pour produire des applications qui répondent à des problématiques dans l'air du temps. En effet, un programmeur habitué à l'environnement .Net sera prêt relativement rapidement et facilement à produire des applications multi-plateformes grâce à .Net MAUI, ce qui peut l'encourager à préférer cette solution à l'apprentissage d'un nouveau langage.

En revanche, bien que les fonctionnalités se rapprochent de celles des frameworks déjà bien installés sur le marché comme React Native et Flutter, il semble assez peu probable que des développeurs en dehors de l'écosystème Microsoft fassent le pas pour y entrer. On peut dire qu'avec .Net MAUI, Microsoft cherche à combler son retard par rapport aux autres géants que sont Facebook et Google, qui soutiennent respectivement React Native et Flutter, pour conserver sa communauté de développeurs.

Est-ce que cette évolution et ce changement de nom suffiront à apporter à .Net MAUI le succès qu'ont pu connaître ses concurrents ? L'avenir nous le dira, mais les éléments techniques sont là. Objectivement, .Net MAUI répond aux critiques qui avaient pû être portées à l'encontre de Xamarin.

Jamstack, sauveur des petits commerces

PARTIE 1

Depuis l'apparition de la COVID-19 et des mesures qui ont été prises pour limiter la pandémie, l'e-commerce s'est développé massivement et a permis aux commerçants de rebondir et de maintenir leur activité. Cependant, lorsque tout le monde se lance dans l'e-commerce, il est important de se différencier des concurrents. Mais quelle technologie choisir ? Quels outils ? Car la réalité du web ne facilite pas le choix. Le Jamstack est une approche pragmatique face aux classiques CMS.

Cynthia Henaff

Cynthia est développeuse front-end autodidacte depuis 2018. Auparavant infirmière de bloc opératoire, elle s'est prise de passion pour le développement et a décidé d'en faire son quotidien. Membre active de la communauté React lilloise, elle est actuellement lead front-end chez Tymate.

Jamstack est une abréviation inventée par Mathias Biilmann, *PDG de Netlify*, qui signifie "JavaScript, API and Markup". Jamstack fait référence à une architecture de développement web qui permet aux développeurs de s'appuyer sur les avantages d'un site web statique tout en conservant les attributs dynamiques d'un CMS orienté base de données, sans la base de données. Elle peut être utilisée pour de nombreux usages : site vitrine (avocat, restaurant…), site personnel (blog, portfolio…), site e-commerce. **Figure 1**

Un site en Jamstack est constitué de différents éléments qui sont :

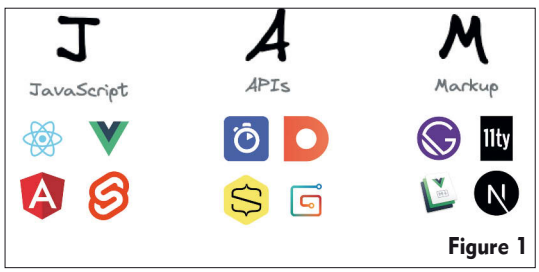


Figure 1

- Un **générateur de site statique** (GSS) Un générateur de site statique est un outil qui construit des pages HTML statiques. À partir du contenu d'un CMS ou d'un autre système, le GSS applique un modèle au choix et génère une structure de fichiers HTML purement statiques, ce qui donne des pages prêtes pour le web. Parmi les plus connus, nous retrouvons *Gatsby*, *Next.js*, ou encore *Eleventy*
- Un **CMS headless** : Un CMS (*Content Management System)* est un type d'application web qui permet de gérer le contenu d'un site Internet. S'il est « headless », alors il expose la donnée sous forme d'API uniquement et est totalement découplé du front-end.
- Des **pipelines d'intégration et de déploiement continu** (CI/CD) : Chaque modification de contenu, sur le CMS ou sur le code source, est notifiée à un serveur d'intégration continue qui va reconstruire le site.
- Un **hébergement CDN** : Un CDN (*Content Delivery Network*) est un moyen d'hébergement depuis plusieurs serveurs situés dans des zones géographiques différentes. Il permet de charger le contenu de la page web depuis le serveur le plus proche de l'utilisateur et donc de réduire le temps de chargement. Les principaux sont *Netlify* et *Vercel*
- Un ou des **services tiers** : L'écosystème florissant des API permet de trouver une solution pour à peu près tous les besoins, de la gestion des paiements et des données de formulaire à l'authentification des utilisateurs, en passant par le stockage et la récupération des données en temps réel et la gestion des tâches de commerce électronique. **Figure 2**

Dans une architecture classique, les pages sont calculées à la volée : chaque requête est reçue par le serveur qui génère la vue et la renvoie à l'utilisateur, alors que dans une architecture Jamstack, les pages sont pré-générées : quand le serveur en reçoit l'ordre, il construit la vue de chaque page du site, et tout le site est servi depuis un CDN.

Pourquoi la Jamstack ?

Il y a beaucoup d'avantages à développer un site de e-com-

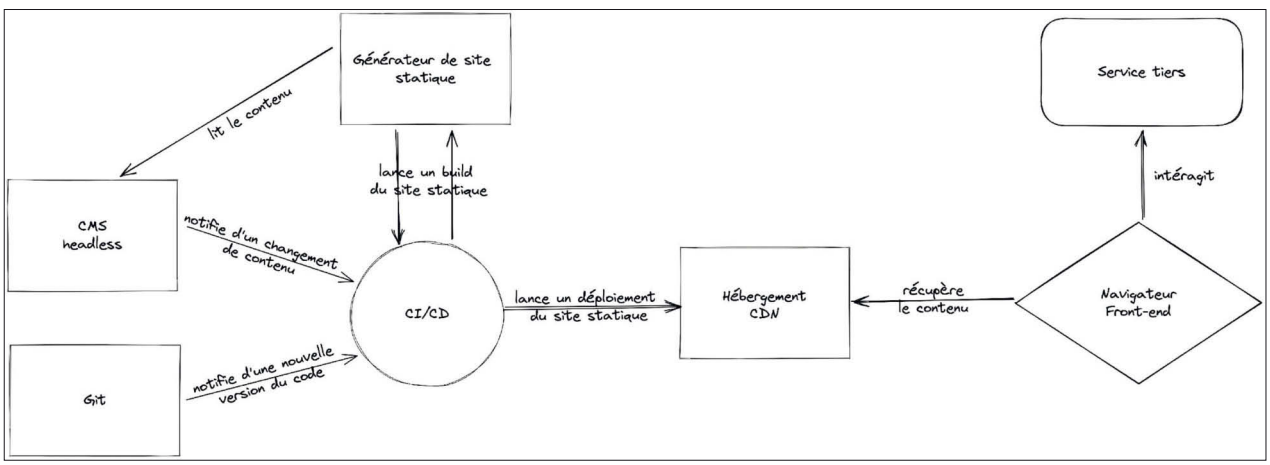


Figure 2

merce avec la Jamstack, mais le plus évident est l'amélioration considérable des performances. Pour commencer, à chaque fois qu'une modification de contenu est appliquée dans le CMS, le site est entièrement reconstruit côté serveur. En conséquence, lorsque les utilisateurs visitent le site, ni les API ni les bases de données ne sont appelées. Si on ajoute à cela que toutes les pages générées sont prêtes à être consommées sur des serveurs de CDN géographiquement proches de l'utilisateur, où qu'il soit dans le monde, alors on obtient des performances nettement accrues par rapport à une architecture classique.

Utiliser un CDN permet également de compenser de façon transparente un éventuel pic soudain de trafic. Il met fin aux problèmes liés à la capacité du serveur ou à une interruption de connexion avec la base de données. De plus, ces pages web statiques pré-rendues sont toujours indexées sans problème par les robots de Google.

Enfin, pouvoir s'appuyer sur une multitude de services tiers et se concentrer exclusivement sur le front-end rend l'expérience de développement bien meilleure. En effet, tous ces services sont développés par des équipes entières consacrées à un seul besoin, à une seule problématique. Cela apporte un vrai gain de temps en termes de développement et d'expérience puisque ces solutions sont déjà optimisées, perfectionnées et sécurisées.

Suite à la mise en production, aucune maintenance n'est nécessaire sur une Jamstack. Avec le CMS headless, le client a le contrôle sur tout le contenu et devient autonome s'il a des modifications à y apporter. Ce CMS a l'avantage d'être simple et intuitif à utiliser pour le client. Ce point en fait également un choix idéal pour les agences et pour les développeurs freelances pour fournir des sites clé en main à leurs clients en les rendant autonomes.

Grâce au CMS et à l'utilisation de services tiers, le coût de développement est réduit par rapport à une architecture classique. Et certains CMS tels que Contentful ou DatoCMS et certains services d'hébergement comme Netlify ou Vercel proposent des plans gratuits, ce qui permet de faire baisser les frais d'hébergement récurrents à presque zéro.

À l'heure actuelle, il existe deux grosses limitations à la solution e-commerce Jamstack. La première se présente en cas de catalogue de produits très important. Par nature avec la Jamstack, lors d'une petite modification sur une seule page, toutes les pages du site sont reconstruites. Si le site contient des milliers de pages, cela peut prendre plusieurs minutes.

Cependant, des solutions existent déjà. Gatsby et NextJS ont publié une option de construction incrémentale. Les constructions incrémentales permettent de ne reconstruire que les pages pertinentes, c'est-à-dire celles qui sont réellement modifiées. Cela permet maintenant d'avoir des sites e-commerce avec de très gros catalogues produits en Jamstack.

La deuxième limitation est le manque de personnalisation du site web pour chaque utilisateur. En effet, le site étant généré en amont (nous sommes dans une approche statique et non dynamique, NDLR), tous les utilisateurs en ont la même version. À l'heure de l'ultra personnalisation, cela peut devenir un vrai problème pour certains. Cependant, le site de commerce en ligne *Everlane* a mis en place une solution très efficace pour ce problème précis [1]. Ils ajoutent des briques dynamiques dans la version statique du site afin de personnaliser le site au profil de l'utilisateur connecté. Si l'utilisateur a déjà commandé des produits féminins, il verra donc mis en avant les articles pour femme. Afin d'éviter la page blanche pendant le temps de chargement de ces briques dynamiques, ils se servent du "Skeleton Screen". Cette technique mise en avant par *Luke Wroblewski* en 2013 part du constat que les pages de chargement habituelles attirent l'attention des utilisateurs sur le fait qu'ils sont en train d'attendre [2]. Le Skeleton Screen consiste à afficher ce à quoi devrait ressembler la page, mais avec des placeholders à la place des données. Ainsi, il permet de faire patienter l'utilisateur en donnant l'impression que le site est chargé. Cette astuce est déjà utilisée par la plupart des géants d'Internet comme Slack, Netflix, ou encore Twitter pour n'en citer que quelques-uns.

La crise COVID et l'avènement de la Jamstack

En mars 2020, le premier confinement dû à la COVID-19 qui devait durer initialement 2 semaines a été mis en place. Malheureusement, la crise sanitaire étant difficilement gérable, nous avons dû enchaîner les confinements. Les victimes économiques de ces confinements à répétition ont été les petits commerces qui, n'étant pas des commerces dits « essentiels », ont dû fermer leurs portes. Malgré leur stock bien rempli, ils étaient dans l'incapacité de vendre et donc de maintenir leur chiffre d'affaires. Le temps était venu pour ces commerces de s'adapter ! C'est là où les solutions telles que le système de click&collect ou les solutions de commerce électronique sont devenues vitales et ont commencé à se développer en flèche.

Quel rapport avec la Jamstack ?

Au-delà de l'élaboration de la stratégie marketing, du choix des produits et du design, il est primordial lors de la conception d'un site de commerce en ligne de choisir avec précaution la technologie qui sera utilisée pour le développer. De plus, de nos jours, 53% de la consommation du web se fait via mobile, il faut donc évidemment penser le site en "mobile first".

Pour un site e-commerce, il est essentiel de disposer d'un site web performant et sécurisé. Nous avons vu précédemment que les Jamstack sont sûres, fiables, flexibles et surtout rapides. Mais pourquoi cela est-il si important ?

Le rapport "*Milliseconds Make Millions*" de Google/Deloitte [3], qui a analysé la corrélation entre les taux de conversion et la vitesse d'un site web, démontre que les deux sont liés : plus les performances sont élevées, plus le taux de conversion augmente. La rapidité joue un rôle essentiel dans la réussite de toute initiative numérique et plus particulièrement pour les sites de commerce en ligne. En effet, il est démontré qu'améliorer le temps de chargement de 0,1 seconde améliore les taux de conversion de 8%. **Figure 3**

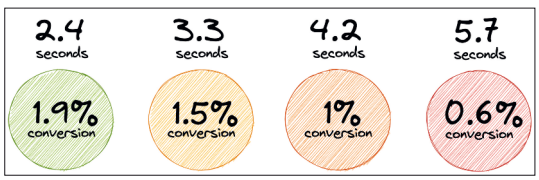


Figure 3

Figure 4

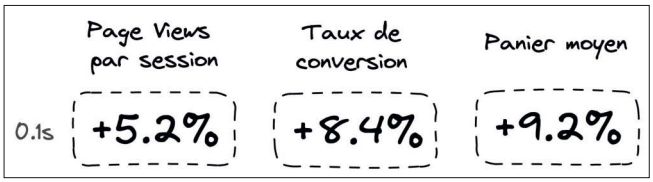
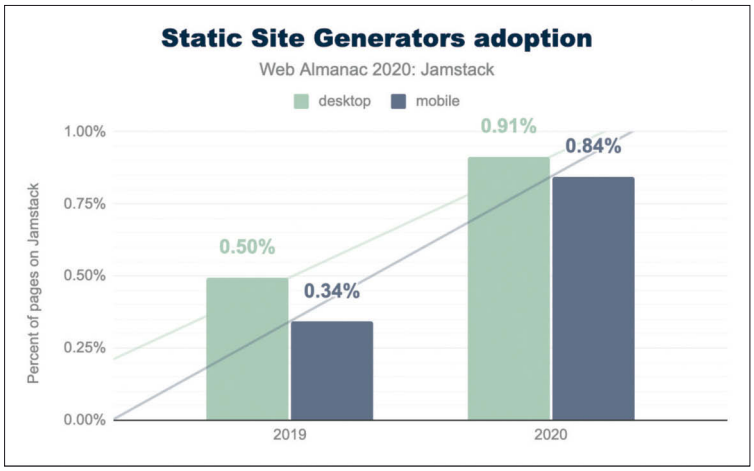


Figure 7

	Desktop	Mobile
2019	0.50%	0.34%
2020	0.91%	0.85%
Évolution	+85%	+147%

Figure 6



Le temps de chargement a également une influence directe sur le taux de rétention. Qui n'a jamais souffert d'une page web trop longue ?

Toujours selon cette étude, 56% des personnes déclarent abandonner le site s'il met plus de 3 secondes à charger et 1 utilisateur sur 5 ne reviendra jamais sur le site en question. Pour présenter cela plus simplement, la moitié des clients quittent un site avant même de l'avoir visité. En plus d'avoir perdu un acheteur potentiel, on perd un client qui ne reviendra jamais.

Lorsque la vitesse d'un site s'améliore de 0,1 seconde, les données montrent une corrélation positive avec le fait que les consommateurs consultent plus de pages par session. Plus de pages consultées par session signifient également plus de transactions et plus de dépenses. En effet, en moyenne les utilisateurs ont dépensé 9,2% de plus. **Figure 4**

Un autre critère primordial à prendre en considération lors du développement d'un site e-commerce est le référencement naturel. Le référencement naturel, ou SEO (*Search Engine Optimization*), englobe toutes les méthodes et techniques qui visent à positionner un site Internet dans les premiers résultats des moteurs de recherche.

La Jamstack offre toutes les qualités en matière de SEO. Un temps de chargement plus rapide est synonyme de meilleur référencement, car les moteurs de recherche favoriseront toujours un site plus léger et plus rapide lors de l'indexation d'un site. Il est évident que plus un site web est haut dans les résultats des moteurs de recherche, plus il sera visible et visité.

Contrairement aux solutions type Magento et Shopify qui permettent de créer facilement des catalogues de produits, une Jamstack donne la possibilité de créer en supplément du contenu éditorial qui est très bénéfique lui aussi pour le référencement naturel. En effet, la *Core Update* de Google de décembre 2020 met dorénavant à l'honneur les sites web avec plus de contenu éditorial.

Les nombreux avantages vus plus haut participent à la hausse de popularité du développement de Jamstack et de sites e-commerce. Cela impacte directement l'écosystème qui voit naître régulièrement de nouveaux outils : à chaque besoin, son service. L'abondance d'outils et de ressources disponibles contribue à repousser les limites de ce qu'il est possible de faire avec un site web statique.

Il est difficile de suivre le développement de l'écosystème tant il est rapide à l'heure actuelle. Cependant, voici une petite liste non exhaustive des principaux services tiers en termes de site e-commerce. **Figure 5**

L'adoption de la Jamstack

Selon les chiffres de 2020, le nombre de Jamstack sur le web a augmenté de 85% par rapport à 2019 pour les sites web desktop et de 147% pour les sites mobiles [4]. Il existe presque deux fois plus de Jamstack qu'en 2019. Cela démontre un vrai signe de l'attrait de la communauté pour cette solution. **Figures 6 et 7**

Selon Forbes, la COVID a accéléré le développement du e-commerce de 4 à 6 ans [5]. Suite à la pandémie, tous les vendeurs physiques lancent leur propre site e-commerce. Il est donc vital de sortir du lot pour optimiser ses revenus. La Jamstack peut faire cette différence. Elle permet d'être en haut des pages de résultats de recherche ce qui apporte un meilleur trafic. Elle garantit des meilleures performances qui influencent l'expérience utilisateur et donc l'association avec la marque.

De plus, opter pour la Jamstack apporte un vrai gain de temps de développement grâce à l'utilisation des services tiers, mais aussi une réduction des coûts par son efficacité et le peu de maintenance nécessaire post développement.

L'émulsion autour de la Jamstack et du e-commerce est tellement importante que Shopify vient de décider de supprimer ses commissions pour le premier million de dollars de revenus annuels [6]. Cela donne la possibilité à beaucoup de commerçants de se lancer dans l'e-commerce : c'est peut-être l'occasion idéale de se lancer ?

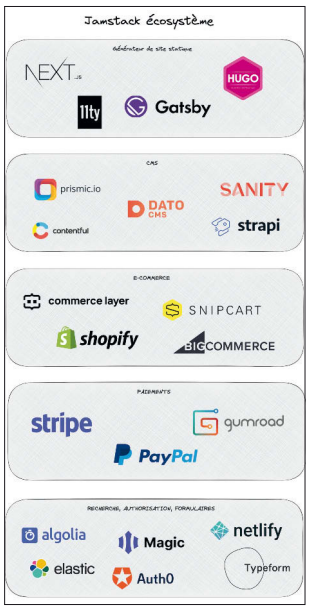


Figure 5

[1] https://www.youtube.com/watch?v=Tjb_oafHVol&t=1s

[2] https://www.lukew.com/ff/entry.asp?1797

[3] https://www2.deloitte.com/ie/en/pages/consulting/articles/milliseconds-make-millions.html

[4] https://almanac.httparchive.org/en/2020/jamstack#fig-2

[5] https://www.forbes.com/sites/johnkoetsier/2020/06/12/covid-19-accelerated-e-commerce-growth-4-to-6-years/

[6] https://www.youtube.com/watch?v=WkkHXluJg_Y

Les anciens numéros de PROGRAMMEZ!

Le magazine des dévs

241

242

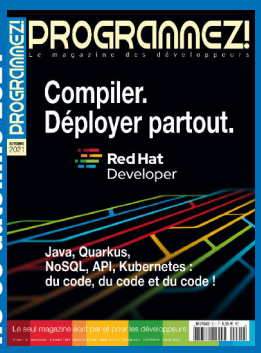
243

HS 02

246

HS 04 été 2021

249



HS 05 automne 2021

250

251

Complétez votre collection....

Tarif unitaire 6,5 € (frais postaux inclus)

☐ *241*	: __ ex	☐ *246*	: __ ex	☐ *250*	: __ ex
☐ *242*	: __ ex	☐ *HS4 été 2021*	: __ ex	☐ *251*	: __ ex
☐ *243*	: __ ex	☐ *249*	: __ ex		
☐ *HS2 automne 2020*	: __ 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 252**

Sylvain Cuenca

Je suis Architecte Système basé en région toulousaine et tout simplement passionné d'informatique depuis mon plus jeune âge, avec bientôt une vingtaine d'années d'expérience dans le monde du logiciel. Les thématiques autour de l'architecture logicielle en général, les performances, la fiabilité, la robustesse, la maintenabilité, l'amélioration continue et l'empreinte mémoire des solutions logicielles sont des sujets que j'aborde pleinement au quotidien. En parallèle, j'effectue beaucoup de veille technologique, m'occupe de la montée en compétence des équipiers et pratique les méthodes japonaises Kaizen sur l'amélioration continue autant sur le plan personnel que professionnel.

Principales nouveautés détaillées de JavaScript de ES2015 (ES6) à ES2021 (ES12)

PARTIE 1 : LES NOUVEAUTÉS DE JAVASCRIPT 2015

Nous allons explorer au travers de cet article les principales évolutions du langage JavaScript de 2015 à 2021 accompagnées d'exemples illustrant les nouveautés apportées par les différentes versions. JavaScript repose sur les standards d'ECMAScript qui lui se charge de définir un ensemble de normes concernant les langages de programmation de type script et standardisées par Ecma International dans le cadre de la spécification ECMA-262.

Historique des versions de ECMAScript

Version	Date de publication	Description
1	Juin 1997	Première édition. Naissance du standard ECMA-262
2	Août 1998	Mise en conformité par rapport au standard international ISO/CEI-16262.
3	Décembre 1999	Expressions rationnelles plus puissantes, amélioration de la manipulation des chaînes de caractères, nouvelles instructions de contrôle, gestion des exceptions avec les instructions try/catch, formatage des nombres. Elle correspond à JavaScript 1.5.
4	Aucune publication	Travail inachevé qui ne verra pas le jour au travers d'une version
5	Décembre 2009 pour la version 5 et Juin 2011 pour la version 5.1	Clarification des ambiguïtés de la 3e version et introduction aux accesseurs, à l'introspection, au contrôle des attributs, aux fonctions de manipulation de tableaux supplémentaires, au support du format JSON et au mode strict pour la vérification des erreurs.
6	Juin 2015	De nombreuses évolutions majeures apportées au langage : fonctions fléchées, portée des variables, classes, modules, promesses pour la programmation asynchrone, déstructuration etc.
7	Juin 2016	Opérateur d'exponentiation, nouvelle méthode pour les prototypes de tableaux
8	Juin 2017	Fonctions asynchrones, mémoire partagée et atomique etc.
9	Juin 2018	Itérateurs asynchrones, clause finally ajoutée à Promise, ajout des notions de propriétés de reste et de décomposition pour l'affectation de la déstructuration des objets etc…
10	Juin 2019	Ajout des fonctions flat et flatMap pour les tableaux, d'une fonction de création d'objets à partir de couple de clé/valeur, ajout de fonctions de suppression de caractères vides complémentaires pour les chaînes de caractères, plus de souplesse pour la clause catch etc.
11	Juin 2020	Nouveau type BigInt, opérateur coalescent des NULL, opérateur de chaînage optionnel, import dynamique, champs privés de classe, nouvelle fonction « allSettled » pour les promesses, ajout de la fonction « matchAll » pour les String, standardisation de l'ordre d'itération sur des propriétés énumérables d'un objet via for..in, standardisation pour l'obtention du contexte global (globalThis), amélioration de l'export des modules nommés, etc.
12	Juin 2021	Séparateur numérique « _ » pour plus de visibilité, fonction replaceAll pour les String, nouvelles fonctions « any » et « race » pour les promesses, opérateurs d'affectation logique, ajout d'accesseurs privés utilisables uniquement dans une classe, etc.

Évolutions du langage

Principales nouveautés de ECMAScript 6 (ES6) ou JavaScript 2015

Nouvelle instruction « let »

La nouvelle instruction « let » permet de déclarer des variables dont la portée est limitée à celle du bloc dans lequel elles sont déclarées. Le mot clé « var », déjà existant, quant à lui permet de définir une variable globale ou locale à une fonction (sans distinction des blocs utilisés dans la fonction). Une autre différence entre « let » et « var » est la façon dont la variable est initialisée. En utilisant « let », la variable est initialisée à l'endroit où le parseur évalue son contenu.

À l'instar de « const », « let » ne crée pas de propriété sur l'objet window quand les variables sont déclarées au niveau global.

Syntaxe :

let variable1 [= value1] [, variable2 [= value2]] [, ..., variableN [= valueN]];

- Les paramètres définis par variable1, variable2, …, variableN doivent être des noms de variable respectant les conventions JavaScript.

- Les valeurs définies par value1, value2, …, valueN sont facultatives ce qui signifie que pour chaque variable déclarée, on peut indiquer, de façon optionnelle, sa valeur initiale. Ces valeurs peuvent être n'importe quelle expression légale.

Exemples :

Illustration de l'utilisation de « let »

```
let code = 'test';
console.log(code);
```

Affichage dans la console :

```
test
```

```
let lastname = 'Durand';
lastname = 'Dupond';
console.log(lastname);
```

Affichage dans la console :

```
Dupond
```

Illustration de la portée des variables via « var » et « let »

```
function varTest() {
  var x = 1;

  if (x === 1) {
    // C'est la même variable x qu'on peut redéclarer
    // dans un autre bloc et sa précédente valeur est écrasée !
    var x = 2;
    console.log(x);
  }

  console.log(x);
}

varTest();
```

Affichage dans la console :

```
2
2
```

```
function letTest() {
  let x = 1;

  if (x === 1) {
    // Ici x est une variable différente et sa portée
    // est limitée au bloc où elle est déclarée
    let x = 2;
    console.log(x);
  }

  console.log(x);
}

letTest();
```

Affichage dans la console :

```
2
1
```

Avis : *A utiliser dès que le besoin s'en fait sentir lorsqu'on peut éviter le mot clé « var ». Je ne conseille pas de redéclarer une variable portant le même nom de variable dans un sous bloc inclus dans un bloc principal, où elle est déjà déclarée, pour des raisons de compréhension de code afin d'éviter toute ambigüité.*

Nouvelle instruction « const »

La nouvelle instruction « const » permet de créer une constante nommée qui est accessible uniquement en lecture. Cela ne signifie pas que la valeur contenue est immuable, uniquement que l'identifiant ne peut pas être réaffecté. Autrement dit, la valeur d'une constante ne peut pas être modifiée par des réaffectations ultérieures. Une constante ne peut pas être déclarée à nouveau.

Les constantes font partie de la portée du bloc (comme les variables définies avec « let ») où elles ont été définies. À la différence des variables définies avec « var », les constantes déclarées au niveau global ne sont pas des propriétés du contexte global. Au sein d'une même portée, il est impossible d'avoir une constante qui partage le même nom qu'une variable ou qu'une fonction.

A noter :

- *Il est nécessaire d'initialiser une constante lors de sa déclaration.*
- *La déclaration « const » crée une référence en lecture seule vers une valeur. Cela ne signifie pas que la valeur référencée ne peut pas être modifiée. Ainsi, si le contenu de la constante est un objet, l'objet lui-même pourra toujours être modifié au niveau de ses propriétés entre autres.*

Syntaxe :

const CONST_NAME_1 = value1 [, CONST_NAME_2 = value2 [, ... [, CONST_NAME_N = valueN]]];

- Les paramètres définis par CONST_NAME_1, CONST_NAME_2, …, CONST_NAME_N peuvent être n'importe quels identifiants valides.
- Les valeurs définies par value1, value2, …, valueN représentent des valeurs associées à des constantes. Ces valeurs peuvent être n'importe quelles expressions valides, y compris une définition de fonction.

Exemples :

Affichage de la valeur d'une constante définie

```
// On définit MY_CONST_1 comme une constante
// et on lui affecte la valeur 1
// Généralement, par convention, les noms des
// constantes s'écrivent en majuscules
const MY_CONST_1 = 1;

console.log('My favorite number is', MY_CONST_1);
```

Affichage dans la console :

```
My favorite number is 1
```

Échec de changement de valeur d'une constante

```
const MY_CONST_2 = 2;

// Cette réaffectation lèvera une exception
// "TypeError: Assignment to constant variable."
MY_CONST_2 = 20;
```

Cela lèvera une exception « TypeError » car on ne peut pas réaffecter une valeur à une constante.

Échec de redéfinition d'une variable via le mot clé « var » au sein d'un même bloc

```
const MY_CONST_3 = 3;

// Cette réaffectation lèvera une exception
// "SyntaxError: Identifier 'MY_CONST_3' has already been declared"
var MY_CONST_3 = 4;
```

Cela lèvera une exception « SyntaxError » car on ne peut pas définir une même variable déjà définie comme étant une constante.

Échec de redéfinition d'une variable via le mot clé « let » au sein d'un même bloc

```
const MY_CONST_4 = 4;

// Cette réaffectation lèvera une exception
// "SyntaxError: Identifier 'MY_CONST_4' has already been declared"
let MY_CONST_4 = 5;
```

Cela lèvera une exception « SyntaxError » car on ne peut pas définir une même variable déjà définie comme étant une constante.

Échec de redéfinition d'une variable via le mot clé « const » au sein d'un même bloc

```
const MY_CONST_5 = 5;

// Cette réaffectation lèvera une exception
// "SyntaxError: Identifier 'MY_CONST_5' has already been declared"
const MY_CONST_5 = 5;
```

Cela lèvera une exception « SyntaxError » car on ne peut pas définir une même variable déjà définie comme étant une constante.

Succès de redéfinition d'une même variable via le mot clé « let » au sein d'un bloc différent

```
const MY_CONST = 10;

// On notera l'importance de la portée du bloc :
if (MY_CONST === 10) {
  // Redéclarer une même variable via let dans
  // un bloc différent fonctionne sans aucun problème
  // et crée une nouvelle variable dans cette portée
  let MY_CONST = 15;

  // Ici, MY_CONST vaut 15
  console.log('My favorite number is', MY_CONST);
}

// Ici, MY_CONST n'a pas changé et vaut toujours 10
console.log('My favorite number is', MY_CONST);
```

Affichage dans la console :

```
My favorite number is 15
My favorite number is 10
```

Succès de redéfinition d'une même variable via le mot clé « const » au sein d'un bloc différent

```
const MY_CONST = 8;

// On notera l'importance de la portée du bloc :
if (MY_CONST === 8) {
  // Redéclarer une même variable via const dans
  // un bloc différent fonctionne sans aucun problème
  // et crée une nouvelle variable dans cette portée
  const MY_CONST = 11;

  // Ici, MY_CONST vaut 11
  console.log('My favorite number is', MY_CONST);
}

// Ici, MY_CONST n'a pas changé et vaut toujours 8
console.log('My favorite number is', MY_CONST);
```

Affichage dans la console :

```
My favorite number is 11
My favorite number is 8
```

Échec de redéfinition d'une même variable via le mot clé « var » au sein d'un bloc différent

```
const MY_CONST = 5;

// On notera l'importance de la portée du bloc :
if (MY_CONST === 5) {
  // L'instruction suivante est remontée dans le
  // contexte global et provoque l'erreur
  // "SyntaxError: Identifier 'MY_CONST' has already been declared"
  var MY_CONST = 10;
}
```

Cela lèvera une exception « SyntaxError » car on ne peut pas définir une même variable déjà définie comme étant une constante via le mot clé « var » étant donné qu'il a une portée globale.

Échec de définition d'une constante sans initialisation par défaut

```
// La définition de const nécessite une initialisation
// L'instruction suivante provoque l'erreur:
// "SyntaxError: Missing initializer in const declaration"
const MY_CONST;
```

Cela lèvera une exception « SyntaxError » car une constante doit toujours être déclarée avec une valeur.

Échec de redéfinition de la valeur d'une constante

```
const MY_CONST = {
  firstname: 'Jean-Claude'
};

// Remplacer la valeur de la constante
```

```
// échouera.
// L'instruction suivante provoque l'erreur:
// "TypeError: Assignment to constant variable."
MY_CONST = {
  lastname: 'Dusse'
};
```

Cela lèvera une exception « TypeError » car on ne peut pas réassigner une valeur à une constante.

Changement des propriétés d'un objet déclaré en tant qu'une constante

```
const MY_DESK = {
  color: 'blue',
  weight: '35'
};

console.log(JSON.stringify(MY_DESK));

// Les clés d'un objet qui est défini en tant que constante
// ne sont pas protégées et on peut donc les supprimer, les
// modifier et en ajouter.
delete MY_DESK.color;
MY_DESK.weight = 25;
MY_DESK.colors = ['blue', 'green', 'yellow'];

console.log(JSON.stringify(MY_DESK));
```

Affichage dans la console :

```
{"color":"blue","weight":"35"}
{"weight":25,"colors":["blue","green","yellow"]}
```

L'exemple précédent montre qu'on peut supprimer, modifier et ajouter des propriétés d'un objet défini en tant qu'une constante.

Ajout de nouveaux items dans un tableau défini comme étant une constante

```
const MY_ARRAY = [];

console.log(MY_ARRAY); // []

MY_ARRAY.push({ country: 'France', language: 'french' });

console.log(JSON.stringify(MY_ARRAY)); // [{"country":"France","language":"french"}]
```

Affichage dans la console :

```
[]
[{"country":"France","language":"french"}]
```

Échec de redéfinition d'un tableau défini comme étant une constante

```
const MY_ARRAY = [];

// Remplacer la valeur de la constante
// échouera.
// L'instruction suivante provoque l'erreur:
// "TypeError: Assignment to constant variable."
MY_ARRAY = [{ country: 'France', language: 'french' }];
```

Cela lèvera une exception « TypeError » car on ne peut pas réassigner une valeur à une constante.

Avis : *A utiliser dès qu'on souhaite définir une constante. Je ne conseille pas de redéclarer une variable portant le même nom de variable dans un sous bloc inclus dans un bloc principal, où elle est déjà déclarée, pour des raisons de compréhension de code afin d'éviter toute ambigüité.*

Ajout du « Spread operator » (Opération de décomposition « ... »)

La syntaxe de décomposition permet d'étendre un itérable (par exemple une expression de tableau ou une chaîne de caractères) en lieu et place de plusieurs arguments (pour les appels de fonctions) ou de plusieurs éléments (pour les littéraux de tableaux) ou de paires clés-valeurs (pour les littéraux d'objets).

Exemples :

Décomposition d'un tableau

```
const sum = (x, y, z) => x + y + z;

const numbers = [2, 3, 5];

console.log('The sum result is:', sum(...numbers));
```

Affichage dans la console :

```
The sum result is: 10
```

Ici on décompose un tableau en récupérant ses éléments présents via l'opérateur « ... » appelé « Spread operator ».

Fusion de deux tableaux

```
const TARN_CITIES = ['Castres', 'Mazamet', 'Albi'];
const HAUTE_GARONNE_CITIES = ['Toulouse', 'Blagnac', 'Castanet',
'Ramonville', 'Balma', 'Labège'];

const MERGED_CITIES = [...TARN_CITIES, ...HAUTE_GARONNE_CITIES];

console.log(MERGED_CITIES);
```

Affichage dans la console :

```
[
 'Castres',   'Mazamet',
 'Albi',      'Toulouse',
 'Blagnac',   'Castanet',
 'Ramonville', 'Balma',
 'Labège'
]
```

Avis : *Très pratique pour faire de la recopie et de la fusion de contenu de tableaux et de la copie d'objets. A utiliser dès qu'on peut et sans limite.*

Définition de Template literals (littéraux de gabarits)

Les gabarits sont délimités par des caractères accent grave (` `) au lieu des apostrophes doubles ou simples. Les gabarits peuvent contenir des espaces réservés (placeholders). Ces espaces sont indiqués par le signe dollar ($) et des accolades (${expression}). Les expressions dans les espaces réservés et le texte compris dans ces espaces sont passés à une fonction.

Pour créer la chaîne finale, la fonction par défaut concatène simplement les différentes parties en une seule chaîne.

Exemples :

Concaténation de chaînes de caractères avant ES6

```
const PERSON = {
  name: 'François Pignon',
  city: 'Toulouse'
};

const MESSAGE = 'Hello ' + PERSON.name + ' from ' + PERSON.city;

console.log(MESSAGE);
```

Affichage dans la console :

```
Hello François Pignon from Toulouse
```

Concaténation de chaînes de caractères à partir de ES6

```
const PERSON = {
  name: 'François Pignon',
  city: 'Toulouse'
};

const MESSAGE = `Hello ${PERSON.name} from ${PERSON.city}`;

console.log(MESSAGE);
```

Affichage dans la console :

```
Hello François Pignon from Toulouse
```

Avis : *Je préconise d'utiliser les littéraux de gabarits car cela permet de rendre plus lisible la concaténation de chaînes de caractères.*

Ajout de la notion de paramètres par défaut pour une fonction

ES6 a introduit un moyen d'ajouter des valeurs par défaut à la liste des paramètres de la fonction. Cela se matérialise par l'ajout du caractère « = » suivi d'une valeur juste après la définition d'un paramètre.

Exemples :

Appel d'une fonction composée de deux paramètres initialisés par défaut

```
const add = (operand1 = 0, operand2 = 0) => operand1 + operand2;

console.log('The addition operation result is:', add());
```

Affichage dans la console :

```
The addition operation result is: 0
```

```
console.log('The addition operation result is:', add(undefined, undefined));
```

Affichage dans la console :

```
The addition operation result is: 0
```

```
console.log('The addition operation result is:', add(null, null));
```

Affichage dans la console :

```
The addition operation result is: 0
```

```
console.log('The addition operation result is:', add(1));
```

Affichage dans la console :

```
The addition operation result is: 1
```

```
console.log('The addition operation result is:', add(1, 4));
```

Affichage dans la console :

```
The addition operation result is: 5
```

Avis : *Très pratique dans certains cas de mettre des valeurs de paramètres par défaut afin d'éviter de devoir coder l'absence de paramètres dans le corps de la fonction. Cela permet de simplifier et de réduire le code. On peut également propager des exceptions à la place de valeurs de paramètres par défaut.*

Ajout du « Destructuring » (l'affectation par décomposition)

Le « Destructuring » est une expression JavaScript qui permet d'extraire des données d'un tableau ou d'un objet grâce à une syntaxe dont la forme ressemble à la structure du tableau ou de l'objet.

Exemples :

Extraction de données depuis un tableau avant ES6

```
const CARTESIAN_COORDINATES = [10, 5, 25];

const X = CARTESIAN_COORDINATES[0];
const Y = CARTESIAN_COORDINATES[1];
const Z = CARTESIAN_COORDINATES[2];

console.log(`The cartesian coordinates are [${X}, ${Y}, ${Z}]`);
```

Affichage dans la console :

```
The cartesian coordinates are [10, 5, 25]
```

Extraction de données depuis un tableau à partir de ES6

```
const CARTESIAN_COORDINATES = [10, 5, 25];

const [X, Y, Z] = CARTESIAN_COORDINATES;

console.log(`The cartesian coordinates are [${X}, ${Y}, ${Z}]`);
```

Affichage dans la console :

```
The cartesian coordinates are [10, 5, 25]
```

Les « [ ] » représentent le tableau à déstructurer et « X », « Y », « Z » représentent les variables où les valeurs du tableau doivent être stockées. Vous n'avez pas besoin de spécifier les index du tableau car ils sont automatiquement implicites. Pendant la déstructuration, certaines valeurs peuvent être ignorées, par exemple la valeur « Y » peut être ignorée comme indiqué ci-dessous :

```
const CARTESIAN_COORDINATES = [10, 5, 25];

const [X, , Z] = CARTESIAN_COORDINATES;

console.log(`The cartesian coordinates are [${X}, ${Z}]`);
```

Affichage dans la console :

```
The cartesian coordinates are [10, 25]
```

Extraction de données depuis un objet avant ES6

```
const MY_CAR = {
  brand: 'Toyota',
  model: 'Yaris',
  color: 'white',
  year: 2021
}

const BRAND = MY_CAR.brand;
const MODEL = MY_CAR.model;
const COLOR = MY_CAR.color;
const YEAR = MY_CAR.year;

console.log(`My car is defined by brand: ${BRAND}, model: ${MODEL},
 color: ${COLOR} and year: ${YEAR}`);
```

Affichage dans la console :

```
My car is defined by brand: Toyota, model: Yaris, color: white and year: 2021
```

Extraction de données depuis un objet à partir de ES6

```
const MY_CAR = {
  brand: 'Toyota',
  model: 'Yaris',
  color: 'white',
  year: 2021
}

const { brand, model, color, year } = MY_CAR;

console.log(`My car is defined by brand: ${brand}, model: ${model},
 color: ${color} and year: ${year}`);
```

Affichage dans la console :

```
My car is defined by brand: Toyota, model: Yaris, color: white and year: 2021
```

Les « { } » représentent l'objet à être décomposé et « brand », « model », « color » et « year » représentent les variables où stocker les propriétés de l'objet. Il n'est pas nécessaire de spécifier la propriété d'où extraire la valeur car l'objet « MY_CAR » contient déjà une propriété appelée « brand » et la valeur est automatiquement stockée dans la variable « brand ».
Comme pour la décomposition de tableau, la décomposition d'objet permet d'extraire uniquement les valeurs nécessaires à un moment donné.

Extraction de la seule propriété « color » de l'objet « MY_CAR »

```
const MY_CAR = {
  brand: 'Toyota',
  model: 'Yaris',
  color: 'white',
  year: 2021
}

const { color } = MY_CAR;

console.log('My car is defined by color:', color);
```

Affichage dans la console :

```
My car is defined by color: white
```

Avis : *Très utile d'extraire partiellement des propriétés d'un objet et des éléments d'un tableau. Cela simplifie notamment l'écriture d'extraction des propriétés d'un objet car on n'a pas besoin de spécifier les chemins complets y accédant. Par exemple, en React, on utilise très souvent le « destructuring » avec « props ».*

Simplification de la définition des propriétés d'un objet « Object literal Shorthand » (Abréviation de littéral d'objet)
ES6 offre une nouvelle façon d'initialiser les objets sans répétition de code, ce qui les rend concis et faciles à lire.
Avant ES6, les objets étaient initialisés en utilisant les mêmes noms de propriétés que les noms de variables qui leur étaient attribués, comme indiqué dans l'exemple ci-dessous :

```
let brand = 'Toyota';
let model = 'Yaris';

const CAR = {
  brand: brand,
  model: model
};

console.log(CAR);
```

Affichage dans la console :

```
{ brand: 'Toyota', model: 'Yaris' }
```

En regardant de près l'exemple précédent, il est clair que « brand:brand » et « model:model » semblent redondants et la nouveauté de ES6 est de supprimer ces noms de variables en double dans les propriétés d'un objet si les propriétés ont le même nom que les variables qui leur sont assignées, comme indiqué dans l'exemple ci-dessous :

```
let brand = 'Toyota';
let model = 'Yaris';

const CAR = {
  brand,
  model
};

console.log(CAR);
```

Affichage dans la console :

```
{ brand: 'Toyota', model: 'Yaris' }
```

Avis : *A utiliser sans limite car cela simplifie le code en enlevant les redondances.*

Ajout d'une nouvelle manière de définir une fonction « Arrow functions » (Fonctions fléchées)
Au travers de cette nouveauté, on a voulu simplifier la syntaxe d'une fonction en introduisant les fonctions fléchées.

Avant ES6, le seul moyen de définir une fonction était de déclarer sa fonction comme suit :

```
function functionName(param1, …, paramN) {
                …
}
```

où « functionName » représente le nom de la fonction et « param1 »,…, « paramN » les paramètres optionnels de la fonction.
Depuis ES6, on a désormais possibilité de déclarer sa fonction comme suit :

```
const functionName = (param1, …, paramN) =>  { … } ;
```

où « functionName » représente le nom de la fonction et « param1 »,…, « paramN » les paramètres optionnels de la fonction.

Exemples :
Définition d'une fonction d'addition avant ES6

```
function addBeforeES6(operand1, operand2) {
  return (operand1 || 0) + (operand2 || 0);
}
```

Définition d'une « Arrow function » d'addition

```
const addFromES6 =
  (operand1 = 0, operand2 = 0) => operand1 + operand2;
```

Définition d'une « Arrow function » qui construit et retourne un nouvel objet

```
const buildCar =
  (brand, model, color, year) => ({ brand, model, color, year });

console.log(JSON.stringify(buildCar('Toyota', 'Mirai', 'white', 2022)));
```

Affichage dans la console :

```
{"brand":"Toyota","model":"Mirai","color":"white","year":2022}
```

Ici les parenthèses ouvrante « (» et fermante «) » dans le bloc de la fonction permettent de protéger la création de l'objet et donc éviter que les accolades ouvrante « { » et fermante « } » ne soient interprétées comme le délimiteur d'un bloc de fonction. Ici, le mot clé « return » est absent dans la fonction mais il est implicite en l'absence du délimiteur de bloc.

Définition d'une « Arrow fonction » avec plusieurs instructions à exécuter

```
const addAndLog =
  (operand1 = 0, operand2 = 0) => {
    const result = operand1 + operand2;

    console.log(result);

    return result;
  };
```

Ici, étant donné que plusieurs instructions définissent le corps de la fonction, elles sont délimitées par des accolades ouvrante « { » et fermante « } » et du moment où on délimite le bloc de la fonction via des accolades, le mot clé « return » devient obligatoire seulement si on souhaite retourner un résultat.

Avis : *Très recommandé car cela simplifie l'écriture des fonctions et les rend moins verbeuses.*

Ajout des classes
ES6 a introduit les classes qui ne sont qu'un « addon syntaxique » par rapport à l'héritage prototypal. Cette nouvelle syntaxe n'introduit pas un nouveau concept d'héritage dans JavaScript et se charge uniquement de fournir une syntaxe plus simple pour créer des objets et manipuler l'héritage. Une classe JavaScript est en fait une fonction spéciale qu'on définit de la même manière qu'une fonction classique par déclaration ou par expression. Leur comportement n'est pas le même que celui des langages de programmation orientés objet tels que Java.

A noter :
ES6 a également introduit deux nouveaux mots-clés, « super » et « extends », qui sont utilisés pour étendre les classes.

Exemples :
Déclaration et instanciation d'une classe simple avec une fonction non statique

```
// Déclaration de la classe Car
class Car {
  constructor(brand, model, year, color) {
    this.brand = brand;
    this.model = model;
    this.year = year;
    this.color = color;
  }

  isFullFilled() {
    return this.brand && this.model && this.year && this.color;
  }
}

const brand = 'Toyota';
const model = 'GR86';
const year = 2022;
const color = 'light grey';

const car = new Car(brand, model, year, color);
console.log(`${JSON.stringify(car)} is${car.isFullFilled() ? '' : ' not'} full filled`);
```

Affichage dans la console :

```
{"brand":"Toyota","model":"GR86","year":2022,"color":"light grey"} is full filled
```

Définition d'une expression de classe anonyme avec une fonction non statique et son instanciation

```
// Déclaration d'une classe anonyme
const AnonymousClassCar = class {
  constructor(brand, model, year, color) {
    this.brand = brand;
    this.model = model;
    this.year = year;
```

```
    this.color = color;
  }

  isFullFilled() {
    return this.brand && this.model && this.year && this.color;
  }
}

const brand = 'Toyota';
const model = 'GR86';
const year = 2022;
const color = 'light grey';

const car = new AnonymousClassCar(brand, model, year, color);
console.log(`${JSON.stringify(car)} is${car.isFullFilled() ? '' : ' not'} full filled`);
```

Affichage dans la console :

```
{"brand":"Toyota","model":"GR86","year":2022,"color":"light grey"} is full filled
```

Définition d'une expression de classe nommée avec une fonction non statique et son instanciation

```
// Expression d'une classe nommée
const NamedClassCar = class Car {
  constructor(brand, model, year, color) {
    this.brand = brand;
    this.model = model;
    this.year = year;
    this.color = color;
  }

  isFullFilled() {
    return this.brand && this.model && this.year && this.color;
  }
}

const brand = 'Toyota';
const model = 'GR86';
const year = 2022;
const color = 'light grey';

const car = new NamedClassCar(brand, model, year, color);
console.log(`${JSON.stringify(car)} is${car.isFullFilled() ? '' : ' not'} full filled`);
```

Affichage dans la console :

```
{"brand":"Toyota","model":"GR86","year":2022,"color":"light grey"} is full filled
```

Définition d'une classe héritant d'une autre classe et son instanciation

```
class Vehicle {
  constructor(brand, model, year, color, wheelNumber) {
    this.brand = brand;
    this.model = model;
    this.year = year;
    this.color = color;
    this.wheelNumber = wheelNumber;
  }
}
```

```
class Car extends Vehicle {
  constructor(brand, model, year, color) {
    super(brand, model, year, color, 4);
  }
}

const brand = 'Toyota';
const model = 'GR86';
const year = 2022;
const color = 'light grey';

const car = new Car(brand, model, year, color);

console.log(car);
```

Affichage dans la console :

```
Car {
 brand: 'Toyota',
 model: 'GR86',
 year: 2022,
 color: 'light grey',
 wheelNumber: 4
}
```

Avis : *Pour la communauté des « Javateux », cela permet de se rapprocher de l'écriture qu'elle connaît bien d'une classe et c'est bien appréciable.*

Ajout des notions d'export/import des modules

Un module ES6 est un fichier contenant du code JavaScript et il n'y a pas de mot clé spécifique désignant un module. Un module est lu comme un script et il peut être défini comme un fichier JS comportant des constantes, des fonctions, des classes etc.

Les modules ES6 sont considérés comme écrits en mode strict. Vous pouvez importer et exporter des modules via les opérateurs « import » et « export ».

Export de modules

Par défaut, dans un module, rien n'est défini comme public ce qui signifie que si d'autres modules veulent bénéficier des fonctions, des constantes, des objets, des classes, etc. qui y sont déclarés, il faut les exporter via le mot clé « export ».

On peut exporter les éléments soit au fur et à mesure soit à la fin du module qui est beaucoup plus recommandé pour plus de clarté et de visibilité.

Pour éviter de désigner par un nom ce qu'on exporte, on peut réaliser jusqu'à un seul export par défaut.

On peut renommer les éléments qu'on exporte via le mot clé « as » suivi d'un caractère espace puis d'un nouveau nom.

Exemples

Export de fonctions nommées, d'une classe nommée, d'une constante renommée et d'une fonction par défaut (fichier « mod.mjs »)

```
const PI = 3.14;

class Vehicle {
  constructor(brand, model, color) {
```

```
    this.brand = brand;
    this.model = model;
    this.color = color;
  }
}

function add(operand1 = 0, operand2 = 0) {
  return operand1 + operand2;
}

function substract(operand1 = 0, operand2 = 0) {
  return operand1 - operand2;
}

export function multiply(operand1 = 0, operand2 = 0) {
  return operand1 * operand2;
}

function log(message) {
  console.log(message);
}

export { add, substract, PI as PI_VALUE, Vehicle };
export default log;
```

Import de modules

L'import consiste à importer des éléments qu'on a bien voulu exporter dans d'autres modules. Pour importer un élément d'un module qui a été exporté en le désignant par un nom, on l'importe en l'encadrant par des accolades en suivant la syntaxe :

« Import { *myItem* } from 'path' ».

Pour importer un élément d'un module externe qui a été exporté par défaut, on n'a pas besoin d'accolade pour importer cet élément et on peut le nommer comme on souhaite en suivant la syntaxe :

« import *myCustomItem* from 'path' ».

Exemples :

Import avec renommage de certains éléments importés

```
import { add as addFunc, substract, multiply, PI_VALUE, Vehicle } from './mod.mjs';

import logFunc from './mod.mjs';

logFunc(addFunc(PI_VALUE, 2));

logFunc(substract(3, 2));

logFunc(multiply(4, 10));

const vehicle = new Vehicle('Toyota', 'Rav4', 'silver');
logFunc(JSON.stringify(vehicle));
```

Affichage dans la console :

```
5.140000000000001
1
40
{"brand":"Toyota","model":"Rav4","color":"silver"}
```

Import multiples

```
import * as math from './mod.mjs';

console.log(math.add(1, 2));

console.log(math.substract(3, 2));
```

Affichage dans la console :

```
3
1
```

Avis : *Les modules sont très pratiques pour centraliser des fonctions utilitaires, des constantes et autres. L'export et l'import des modules contribuent à une meilleure structuration de son code architecturalement parlant.*

Ajout des Promise (promesses)

JavaScript a la volonté de se tourner de plus en plus vers des modèles asynchrones car cela permet au moteur JavaScript de gérer plusieurs tâches en parallèle (événements, affichage, interrogation sur le local store…) et de conserver une interface réactive malgré le fait qu'il soit « Single Thread ».
Pour formaliser cela, ES6 a créé un objet nommé Promise (Promesse). Les promesses sont donc des objets qui retournent la valeur d'une opération asynchrone, autrement dit elles représentent une valeur future.
Elles disposent de méthodes permettant de traiter le résultat une fois l'opération accomplie (then() et catch()) et vont nous permettre de nous affranchir des « callback » des fonctions, qui sont désormais attachées à la promesse. Cela permet de nous libérer de l'inversion de contrôle induit par les callbacks (c'est donc la fonction appelée qui est chargée de les lancer).
On peut chaîner les promesses via l'opération « .then() ».
Pour rappel, une « callback » est une fonction de rappel passée dans une autre fonction en tant qu'argument, qui est exécutée à l'intérieur de la fonction externe pour accomplir une action.
Une promesse comporte 3 états possibles :

- Fullfilled (résolue) : la valeur de la promesse est retournée sans erreur. La fonction « resolve » est appelée.
- Rejected (rejetée) : Une erreur est survenue et on a possibilité d'effectuer un traitement via la fonction « reject ».
- Pending (en cours) : La valeur retournée par la promesse n'est pas encore disponible.

A noter : *une fois la promesse résolue ou rejetée, elle ne peut plus changer d'état. Il existe une fonction « Promise.all » qui prend un tableau de promesses en paramètre et est résolue lorsque toutes les promesses sont résolues ou si l'une d'entre elles est rejetée. Si aucune promesse n'a échoué, elle renvoie un tableau qui contient les résultats des promesses en conservant l'ordre des promesses défini dans le tableau initial passé en paramètre.*

Exemples :

Définition d'une fonction asynchrone permettant d'attendre le temps défini par l'utilisateur et d'afficher le résultat une fois la promesse résolue avec succès

```
const wait =
  (timeToWait = 0) => new Promise((resolve) => setTimeout(
    () => resolve(`Promise resolved after ${timeToWait}ms`), timeToWait));

wait(3000)
  .then(
    (result) => console.log(result))
  .catch(error => console.error(error));
```

Affichage dans la console :

```
Promise resolved after 3000ms
```

Chaînage de deux promesses

```
const wait =
  (timeToWait = 0) => new Promise((resolve) => setTimeout(
    () => resolve(`Promise resolved after ${timeToWait}ms`), timeToWait));

wait(3000)
  .then(
    (result) => {
      console.log(result);
      return wait(2000);
    })
  .then((result) => console.log(result))
  .catch(error => console.error(error));
```

Affichage dans la console après 3 secondes d'attente :

```
Promise resolved after 3000ms
```

Affichage dans la console après 2 secondes supplémentaires d'attente :

```
Promise resolved after 2000ms
```

Echec d'une promesse

```
const rejectedFunc =
  () => new Promise(
    (resolve, reject) => reject('Promise is rejected'));

rejectedFunc()
  .then(
    () => console.log('All is OK'))
  .catch((error) => console.error(error));
```

Affichage dans la console :

```
Promise is rejected
```

Exécution avec succès de deux promesses en parallèle

```
const wait =
  (timeToWait = 0) => new Promise((resolve) => setTimeout(
    () => resolve(`Promise resolved after ${timeToWait}ms`), timeToWait));

Promise.all([wait(3000), wait(2000)])
  .then((result) => console.log(result))
  .catch((error) => console.error(error));
```

Affichage dans la console après 3 secondes d'attente :

```
[ 'Promise resolved after 3000ms', 'Promise resolved after 2000ms' ]
```

Exécution de deux promesses en parallèle dont une qui échouera

```
const waitWithSuccess =
  (timeToWait = 0) => new Promise((resolve) => setTimeout(
    () => resolve(`Promise resolved after ${timeToWait}ms`), timeToWait));

const waitWithFailure =
  (timeToWait = 0) => new Promise((resolve, reject) => setTimeout(
    () => reject(`Promise rejected after ${timeToWait}ms`), timeToWait));

Promise.all([waitWithSuccess(2000), waitWithFailure(3000)])
  .then((result) => console.log(result))
  .catch((error) => console.error(error));
```

Affichage dans la console après 3 secondes d'attente :

```
Promise rejected after 3000ms
```

Avis : *A utiliser sans modération dès qu'on souhaite coder en asynchrone.*

Dans la partie 2 à paraître dans Programmez! 253, nous explorerons les nouveautés d'ECMAScript 7 alias JavaScript 2016.

Jean-Paul Gallant

Practice Manager chez Altran (aujourd'hui Capgemini-Sogeti) depuis 2005. Dépose à l'INPI des méthodes et modes opératoires de stratégie de recette et de qualification des applications informatiques.

Craftsmanship, Clean Code, retour d'expérience

La traduction exacte de **Software Craftsmanship** est **« artisanat du logiciel ».** Comme son nom l'indique nous verrons en quoi le logiciel peut s'apparenter à de l'artisanat et quelle est la valeur qui s'en dégage. Le Software Craftsmanship s'inscrit à la suite d'une lignée concernant l'évolution de la pensée agile démarrée en 1986.

Il s'agissait alors de la Spirale de Bohem et de l'itératif incrémental. Plus tard en 1991 apparaissait le RAD de J.Martin que l'on peut classifier comme le père de la méthode Agile. De 1994 à 2000 émergeaient (US) XP, (US) Scrum, (US) FDD, (US) ASD, (US) Crystal. A cette même période le RAD était importé en France par Jean-Pierre Vickoff.
En 2001 sortait le Manifeste Agile. En 2009 les pratiques Agile atteignaient une certaine maturité et le Répertoire Agile Alliance apparaissait.

De 2010 à nos jours le Software Craftsmanship porté par **Robert C Martin (Oncle Bob)** et faisant l'objet du présent article, s'introduit avec le **Clean Code** dans la lignée des méthodes et techniques de type **DEVOPS** : l'intégration continue, **l'ATDD** : le développement piloté par l'acceptance test, le **TDD** : le développement piloté par les tests et le **BDD** : le développement piloté par le comportement applicatif.

Nous commencerons par la philosophie du Software Craftsmanship. Il s'agit de l'artisanat du logiciel, c'est-à-dire la façon même dont est conçu et écrit le code. **Comme un artisan, le développeur mène la pratique de son expérience avec un grand savoir-faire aboutissant à l'excellence.**

Une pizza c'est assez simple à faire :
- Faisable chez soi,
- De très nombreux restaurants en proposent.

Une bonne pizza c'est assez compliqué à faire :
- On a tous notre restaurant préféré pour ses bonnes pizzas.

Une très bonne pizza c'est très rare et demande un grand savoir-faire.
- On est dans ce cas dans le **Software Craftsmanship : l'excellence**

Cette excellence trouve une complémentarité dans le mode Agile et la souplesse des cycles de développement (1). Les ingrédients nécessaires pour produire un logiciel de qualité sont l'association de la **Qualité, de l'humilité, du partage, du pragmatisme et du professionnalisme**.

Mais il ne s'agit pas de vains mots. **La Qualité** c'est la conception simple, le clean code et le refactoring des tests dont le TDD. **L'humilité** c'est « Je me remets en question et je m'améliore en continu » (rétrospectives de Scrum). **Le partage** c'est « Je programme en groupe » : propriété collective du code. **Le pragmatisme** c'est « Je comprends les contraintes et m'adapte si nécessaire » (rétrospectives de *Scrum*). **Le professionnalisme** c'est « Je traite mon client comme un partenaire » (*principe du « courage » d'XP Extreme programing*).

Ceci nous amène au **Manifeste du Software Craftsmanship** (2)
- Pas seulement des logiciels opérationnels, mais aussi des logiciels bien conçus.
- Pas seulement l'adaptation aux changements, mais aussi l'ajout constant de valeur.
- Pas seulement les individus et leurs interactions mais aussi une communauté professionnelle.
- Pas seulement la collaboration avec les clients, mais aussi des partenariats productifs.

Le Clean Code

Les règles du Clean Code

- Chaque ingrédient à sa place
- Les ingrédients sont bons…

Les règles du Clean Code peuvent s'énoncer de la façon suivante :

• Tu es responsable de ton code

Tu es responsable de la qualité de ton code, tu ne dois pas rejeter ta responsabilité sur les autres.

• Nommage des variables

Elles doivent représenter ce qui leur correspond, elles doivent être prononçables et pouvant être recherchées. Évite les noms compliqués.

• Règles sur les fonctions

Rédige le nom des fonctions avec un verbe à l'infinitif (créer…). L'objectif est que le nom de la fonction puisse te donner suffisamment d'informations pour t'éviter d'aller lire

le code. Une fonction doit faire une seule chose. Son code idéal doit être séquentiel, lisible de haut en bas. Le nombre de ses paramètres doit être réduit (éviter plus de 3). Évite les duplications de fonctions.

• **Du code explicite à la place de commentaires**
Il faut des commentaires cependant, mais uniquement lorsque c'est nécessaire, surtout pour aider à la compréhension. Lorsque le code est suffisamment explicite, il n'y a pas besoin de commentaire…

• **Les règles sur les commentaires**
Bons commentaires : Être informatifs, pour clarifier le code. Pour donner des avertissements. Pour générer automatiquement de la documentation (Ex.: Java Doc).
Mauvais commentaires : Non nécessaires, redondants, trompeurs (pas en phase avec le code). Évite les gros séparateurs de code, les commentaires de fin de bloc. Évite les html dans le code. Évite l'historique (les gestionnaires de code savent le faire).

• **Du code lisible plutôt que du code optimisé** (3)
Écrit plutôt du code non optimisé mais lisible et compréhensible au lieu du code optimisé non compréhensible et non lisible. Le rapport entre le temps de lecture et d'écriture du code est environ de 10 pour 1. Donc rendre le code plus facile à lire, le rend plus facile à le maintenir et à écrire le nouveau code.

• **Les tests unitaires**
Apprend à maîtriser l'approche TDD (le pilotage du développement par les tests). Fait des tests unitaires logiques et nécessaires. Sache créer le lien entre le BDD (le pilotage par le comportement applicatif) et le TDD.

• **Utilise des Linters**
Utilise l'analyse statique de code. Ces outils produisent des warnings en fonction de règles (exemple SONAR). Objectif : améliorer ton code. Par contre sache où mettre le curseur pour t'améliorer (régler les règles).

• **Itère sur la revue de code**
Revoit les fonctionnalités écrites, reviens-y et fait de la revue de code avec des collègues. Vérifie qu'ils comprennent bien ton code. Les règles doivent être communes pour tous tes collègues développeurs

• **Pratique, pratique, pratique le Clean code**
C'est un état d'esprit. Il faut que tu sois toujours dans l'amélioration de ton code. Tu dois débattre de ton code avec tes collègues. Les règles ne doivent pas être figées mais peuvent évoluer.

L'ATDD, le BDD et le TDD

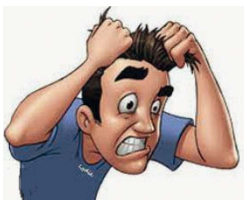
- Travaille sur la pâte
- Goûte les ingrédients avant de les installer
- Conçoit le goût voulu en fonction des préférences des invités…

Le **TDD** consiste à mettre le test en amont du développement. Il s'agit du « Test Driven Development » ou « pilotage du développement par les tests ». **Avant de développer on doit tester … mais comment ça peut marcher si on n'a pas encore le code ?**

Là, intervient le **BDD** : « Behavior Driven Development » ou « le pilotage du développement par le comportement applicatif »
De quoi s'agit-il ?
Avant de commencer à naviguer quelque part, il vaut mieux savoir où aller ! (4)
La nouvelle fonctionnalité à développer indique le but. L'**ATDD** (Acceptance Test Driven Development) crée « les routes possibles de navigation » en livrant une vision globale des parcours utilisateurs.

Il s'agit ainsi de collaborer avec les personnes impliquées sur le projet (Product Owner, testeur…) : Comprendre la vision qu'elles ont de la future fonctionnalité et confirmer qu'il y a un vrai besoin pour le changement proposé.
A partir de là, il est important de définir les choses à tester… et on pourra utiliser les principes du BDD. Il s'agira donc d'utiliser la syntaxe suivante :
Étant donné : **Given;** Quand : **When,** Alors : **Then;** Et : **And**

Exemple :
Given Un utilisateur laisse un commentaire
When le commentaire dépasse 1000 caractères
Then le commentaire ne doit pas être sauvegardé
And l'utilisateur doit voir un message d'erreur



Mais comment tester ?? Il n'y a pas de code !!!

Le test va échouer parce qu'il n'y a pas de code associé avec ! C'est comme si vous aviez le texte sur un côté de la page et que vous deviez répondre aux questions de compréhension sur l'autre côté.
Afin de passer le test, vous allez écrire le code dans un autre fichier qui évaluera la longueur du commentaire et renverra une erreur (ou pas).
Lorsque le code est écrit et que le test est réussi, vous passez au prochain test et donc au prochain code.

L'échec est bénéfique. Tous vos tests doivent échouer au début !
On arrive ainsi au cycle : Rouge-vert-refactor

Rouge : Vous écrivez un test simple qui échouera avant même d'avoir écrit le code. Donc le test échoue obligatoirement puisque le code n'est pas encore écrit. Le test est donc rouge.
Vert : Vous écrivez le code le plus simple possible pour faire passer le test. Donc le test réussit, il est vert.
Refactor : le code que vous avez écrit pour faire passer le test est peut-être trop simple. Un refactor de ce code est maintenant possible.

Mon retour d'expérience concernant le Craftsmanship et le Clean Code

J'ai moi-même suivi les principes et les règles du Craftsmanship et du Clean Code édictées dans ces chapitres. Bien que mon domaine d'activité ne soit pas le développement étant practice manager dans le domaine du test chez Altran depuis très longtemps (j'ai cependant une longue expérience professionnelle du développement), j'ai pu en retirer les expériences suivantes.

Le contexte

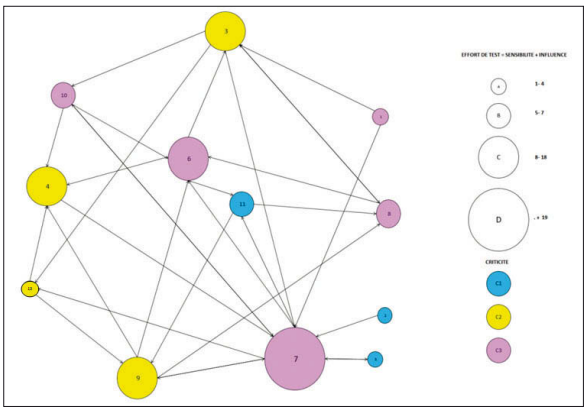
Nous avons, chez Altran des méthodes très affinées dans le domaine du test et parmi ces méthodes nous avons la construction des **matrices de couvertures des exigences métier et des tests** en suivant des modes opératoires très précis dans le cadre d'ateliers fonctionnels mis en œuvres avec les sachant(s) métier de nos clients (voir chapitre suivant : « Notre perception du CRAFTSMANSHIP dans le test / Le cadre méthodologique », voir également mon article dans le Livre Blanc du CFTL Edition 2021 : « Automatisation des activités de test » / Gestion industrialisée et automatisée de l'effort de test ») (5). Ces matrices de couvertures, nous les élaborons dans un outil développé simplement sous Excel qui fonctionne en liaison avec les outils du marché de gestion des tests (ALM, Squash…). **Cette méthodologie est aussi bien mise en œuvre dans les projets « cycle en V » qu'en mode Agile.**

Sur ces matrices nous lions les exigences métier avec les cas d'utilisations. Les cas de test potentiellement dans les cellules à l'intersection des exigences et des cas d'utilisations, sont définis ultérieurement. Nous identifions sur ces matrices les **impacts fonctionnels ainsi que l'effort de test** étant noté par le nombre d'impacts associés à chaque exigence. **Figure 1**

On obtient ainsi une valeur de « sensibilité » et « d'influence ». On dira qu'une exigence est d'autant plus sensible qu'elle est impactée par les autres exigences, on dira qu'elle est d'autant plus influente qu'elle impacte elle-même les autres exigences.

Ces valeurs nous permettent d'obtenir la valeur **de l'effort de test nécessaire et de prioriser ainsi les cas de test et les scénarios de test.** Notre outil élabore ainsi, à partir du contenu de la matrice, à l'issue de l'atelier, une **matrice de régression** calculant automatiquement les valeurs de **sensibilité, d'influence et d'effort de test par exigence**. Bien d'autres fonctionnalités permettent notamment, lorsqu'on évalue la complexité et le nombre de cas de test par cellule de la matrice, de générer les cas de test non valorisés et de les importer directement sous un gestionnaire de test avec l'architecture fonctionnelle modélisée dans la matrice. **Figure 2**

Cependant il nous manquait une vision directe et plus immédiate de cette architecture sous une autre forme que la vue matricielle.



J'ai donc, à ce moment-là, décidé de **développer sous Excel une vue cartographique** en modélisant les exigences par des cercles dont leur dimension est en fonction de l'effort de test et la couleur leur criticité. Chacun de ces cercles (ou exigence) est relié aux autres cercles par des vecteurs indiquant l'impact entre ces autres exigences et l'orientation de ces vecteurs donnant le sens de l'impact.

Pour développer le logiciel (en VBA sous Excel) **j'ai décidé d'appliquer tous les principes et règles du Craftsmanship et du Clean Code.**

Mon retour d'expérience

Mon premier constat a été que je n'ai **pas mis plus de temps** que ce que j'aurais pu mettre à développer ce logiciel si je n'avais pas suivi les principes du Craftsmanship et du Clean Code. Peut-être même ais-je mis moins de temps à le développer.

Figure 1

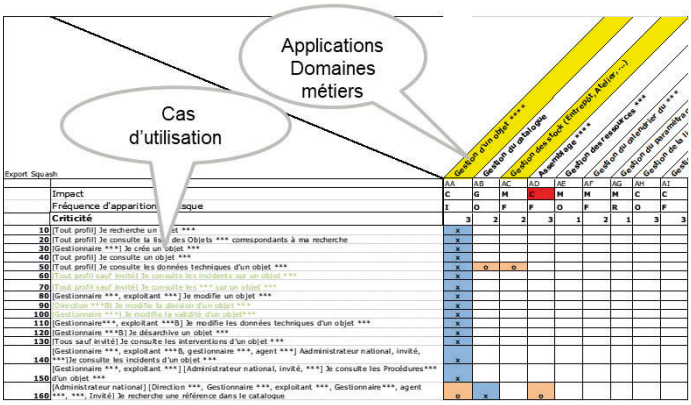
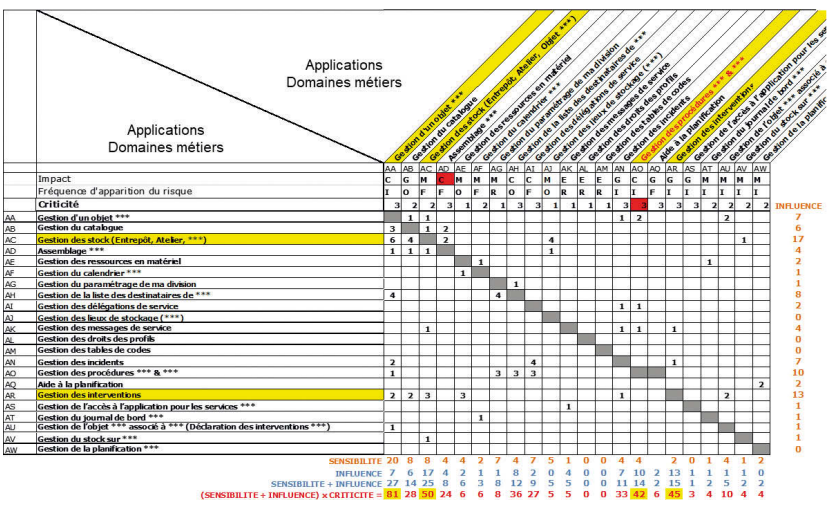


Figure 2



J'ai donc démarré le développement en utilisant le principe du **TDD c'est-à-dire du test avant le développement**. Cela peut être intéressant du point de vue philosophique car la première question que je me posais était la suivante : comment je peux tester sans avoir codé la génération automatisée d'objets graphiques sous la forme de cercles sur une feuille Excel ?
Car le test, à ce niveau, peut être simplement visuel. Donc j'ouvre une feuille Excel vierge et je ne vois rien, pas un seul cercle, **donc mon test a échoué et alors ?**
Cela veut dire qu'avant de coder la génération de cercles sur ma feuille **il faut que je teste s'il existe un ou plusieurs cercles sur cette feuille.**

Un second principe vient alors à la rencontre du premier. Mais pourquoi tester s'il existe des cercles puisque, **si je les crée, je n'ai donc pas besoin de vérifier leur existence**. C'est là toute la subtilité du problème. En effet, si la vérification de leur existence ne m'était pas parue immédiate au départ, elle m'a permis par la suite de **maîtriser le contenu de l'ensemble de ces objets graphiques** sachant qu'il fallait tous les gérer et les relier les uns aux autres (un objet étant relié à n autres) en fonction des données utilisées sur les impacts fonctionnels.

Cette option qui consiste à vérifier l'existence des objets graphiques **s'est avérée ainsi forte utile pour la poursuite du développement** et m'a permis d'éviter pas mal de problèmes ultérieurs.

En suivant les règles du Clean Code j'ai donné des noms de fonctions relativement longs pour qu'ils **représentent une information claire** (« Rédige le nom des fonctions avec un verbe à l'infinitif (créer…). L'objectif est que le nom de la fonction puisse te donner suffisamment d'informations pour t'éviter d'aller lire le code » : 3e règle du Clean Code, voir chapitre « Les règles du Clean Code »). Cela m'a évité pas mal de retours arrière dans la compréhension de l'objet des différentes fonctions.

Je suis parti également sur la **différenciation des fonctionnalités que j'utilisais en tant que TDD et en tant que BDD** (souvenez-vous « C'est comme si vous aviez le texte sur un côté de la page et que vous deviez répondre aux questions de compréhension sur l'autre côté. Afin de passer le test, vous allez écrire le code dans un autre fichier qui évaluera la longueur du commentaire et renverra une erreur, ou pas », voir au chapitre : « L'ATDD, le BDD et le TDD »).

J'ai donc différencié ces deux types de fonctions en les faisant précéder d'un verbe à l'infinitif pour les fonctions de type BDD et des caractères « TEST » pour les fonctions de type TDD. Au bout d'un certain temps, il a été difficile de décider si une fonction était de type TDD ou BDD. En fait, souvent les fonctions TDD évoluaient vers des fonctions BDD. Mais le fait de **conserver et d'utiliser pendant tout le développement les strates plus ou moins anciennes des fonctions TDD** permettait d'éviter facilement les régressions éventuelles en testant facilement.

L'application systématique des règles du Clean Code (je n'évoquerai pas mon retour sur toutes les règles que j'ai suivi) m'a permis de **produire un développement solide, très stable** pourtant très fourni en analyse mathématique, notamment les fonctions trigonométriques concernant par exemple les choix des connecteurs sur les cercles lorsqu'un vecteur doit s'y être connecté en fonction de sa direction de départ, l'interdiction pour un cercle d'en recouvrir un autre, pour un vecteur de recouvrir un cercle, le positionnement des cercles etc.

CODING DOJO – KATA

Le ***coding dojo*** (6,7) est une rencontre entre plusieurs personnes qui souhaitent travailler sur un défi de programmation de façon collective. Le défi peut être un problème algorithmique à résoudre ou un besoin à implémenter.
Chaque *coding dojo* se concentre sur un sujet particulier, et représente l'objectif de la séance. Ce sujet doit permettre d'apprendre de façon collective sur le plan technique et sur la manière de réussir le défi.

KATA : Une personne démontre au reste du groupe à partir de zéro comment résoudre le problème. Un kata est aussi un exercice de programmation que l'on répète pour perfectionner ses compétences.

Prendre un petit problème de programmation et **essayer de mettre en œuvre une solution encore et encore**. La solution réelle est moins importante que l'acte de résoudre le problème. Les petites techniques de résolution de problèmes font progressivement partie de la mémoire musculaire, **un peu comme une répétition de l'entraînement sportif.**

Notre perception du CRAFTSMANSHIP dans le test

Un trio dans le test

La méthode **« des trois amigos »** (8) concerne la mise en œuvre d'un trio dans le test.
Il s'agit du Product Owner (PO), du développeur et du testeur :

- Le **Product Owner** (PO) priorise et rédige les **user stories** (US)
- Le testeur et le développeur réalisent une revue des US. Le testeur les complète en formalisant des BDD (se renvoyer aux BDD)
- Le développeur développe les US, réalise les tests unitaires et le **testeur anime les séances de testing**
- Le testeur teste les US, **automatise les tests au fil de l'eau** pour constituer progressivement un **référentiel de tests de régression** permettant de garantir une **couverture de test suffisante dans la durée**
- Le testeur **développe les bouchons, les outils de test et fournit les jeux de données nécessaires**

Les testeurs positionnés dans les équipes agiles (squads) ont en général des compétences en développement.

Travail en équipe et Co-localisée

Les testeurs interviennent tout le long du cycle projet, pour :

- Participer à toutes les cérémonies : Sprint Planning,

Meeting, Démo, Rétrospective dans lesquelles les caractéristiques du produit sont présentées, analysées ou estimées.
- S'assurer que les niveaux de qualité de l'application souhaités soient atteints.
- Un formalisme de User Stories (langage Gherkin)
- Mettre en place une stratégie de test/développement
- Proposer la démarche d'automatisation des tests.

Un Rôle accru du testeur

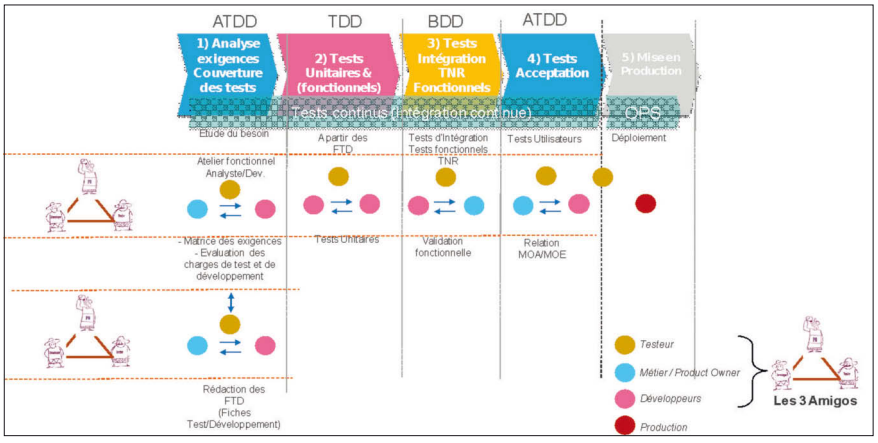
Le rôle du testeur consiste à :
- Partager la vision du produit,
- Collaborer avec le métier pour affiner les user stories et les critères d'acceptation,
- Définir la stratégie et la charge de test, pilote et/ou réalise la conception et l'exécution des tests,
- Déterminer les cas de tests à automatiser et réalise les scripts d'automatisation,
- Communiquer sur le niveau de qualité de l'application.

Quels sont les avantages ?

Les avantages des trois Amigos sont les suivants :
- **Détecter le plus rapidement** possible les bogues d'intégration dans le cadre d'un développement collaboratif,
- **Réduire les écarts** entre le business et le développement logiciel en se recentrant sur les besoins business par la mise en place de nouvelles méthodes mieux adaptées utilisant les outils appropriés.
- Positionner différemment **le test**. Il doit se situer au niveau du besoin **totalement en amont du développement.**
- **Décloisonner** la MOA, le développement et la production pour travailler de manière collaborative, plus rapidement et efficacement, en améliorant la qualité, la stabilité et la sécurité de la production.

Le cadre méthodologique

Le schéma ci-dessous représente le cadre méthodologique dans lequel on peut se situer pour parvenir à l'intégration continue ou au déploiement continu (DevOps).



L'ATDD peut se mettre en œuvre dans le cadre de l'analyse des exigences métier en **début de traitement ou dans le backlog de produit en mode Agile**. On peut, tout à fait, dans ce cadre, organiser des ateliers fonctionnels où vont participer les sachants métiers.

Les 3 Amigos constituent préalablement une couverture des tests avant le développement dans une matrice de couverture des exigences.

Le TDD peut ensuite être mis en œuvre par les développeurs **en relation avec les deux autres partenaires Amigos** (le/les testeur(s) et le Product Owner).

C'est toujours un **travail du développeur en relation avec les deux autres partenaires Amigos** que de mettre en œuvre le BDD en amont et en aval du TDD pour analyser le comportement applicatif et définir les tests nécessaires au développement (comme vu précédemment) en définissant les fonctionnalités de bas niveau qui vont permettre de tester, ces fonctionnalités associées à des mots clés qui vont les activer. Ces mots clés mis bout à bout peuvent permettre d'exécuter des tests automatisés **à partir de phrases simples mais significatives pour des fonctionnels non techniques.**

En ATDD, on écrit le test d'acceptance avant le code au même titre qu'un test unitaire pour le TDD.

Une bonne maîtrise du **CRAFTSMANSHIP et du Clean Code,** permet la mise en œuvre de **l'intégration continue ou DevOps** dans une chaîne entièrement automatisée du développement jusqu'à la mise en production.

Références

(1) Le testeur Agile – La taverne du testeur
https://latavernedutesteur.fr/2021/06/07/le-testeur-agile/

(2) C'est quoi le Software Craftsmanship ? (linkedin.com)
https://fr.linkedin.com/pulse/cest-quoi-le-software-craftsmanship-houssam-tiouant

(3) 10 bonnes pratiques pour du code lisible | Spiria
https://www.spiria.com/fr/blogue/methodes-et-bonnes-pratiques/10-bonnes-pratiques-pour-du-code-lisible/

(4) Apprenez le behavior driven development (BDD) - Testez l'interface de votre site - OpenClassrooms
https://openclassrooms.com/fr/courses/3504461-testez-linterface-de-votre-site/4270566-apprenez-le-behavior-driven-development-bdd

(5) Livre Blanc du CFTL Edition 2021 : « Automatisation des activités de test » / Gestion industrialisée et automatisée de l'effort de test » J-P GALLANT

(6) Coding dojo — Wikipédia (wikipedia.org)
https://fr.wikipedia.org/wiki/Coding_dojo

(7) L'univers des coding dojos - Blog (codeinsider.fr)
https://blog.codeinsider.fr/lunivers-des-coding-dojos/

(8) Les 3 amigos en agile - My Agile Partner Scrum
https://blog.myagilepartner.fr/index.php/2019/02/06/les-3-amigos-agile/

MongoDB & Java: Client-Side Field Level Encryption.

Le Client-Side Field Level Encryption (CSFLE) est une nouvelle fonctionnalité ajoutée dans MongoDB 4.2 qui vous permet de chiffrer certains champs de vos documents MongoDB avant de les transmettre via le réseau au cluster pour les stocker.
C'est la sécurité ultime contre tout type d'intrusion ou d'espionnage autour de votre cluster MongoDB. Seule l'application avec les bonnes clés de déchiffrement peut déchiffrer et lire les données protégées.

Maxime Beugnet

Senior Developer Advocate @MongoDB

Maxime travaille avec MongoDB depuis 7 ans et dans l'industrie informatique depuis 10 ans.

Il est formateur MongoDB et Java, avec les certifications DEV et DBA MongoDB. Maxime a rejoint MongoDB en tant que Developer Advocate il y a 4 ans pour partager son expérience avec la communauté. Il aime le code propre, la plongée sous-marine, les compétitions de programmation et les vikings !

Instructions de Mise en Place

Tout le code présenté dans cet article est disponible dans le dépôt Github que vous pouvez cloner librement :
https://github.com/mongodb-developer/java-quick-start

Si vous n'avez pas encore configuré votre cluster gratuit sur MongoDB Atlas, c'est le bon moment pour le faire.
Vous avez toutes les instructions sur ce site:
https://docs.atlas.mongodb.com/getting-started/.

Dans cet article, je vais vous présenter l'API Java CSFLE et la logique de fonctionnement de la cryptographie à travers un exemple simple. Je tiens aussi à préciser que seule la version communautaire de MongoDB est nécessaire pour bénéficier de cette fonctionnalité telle que présentée ici. Cependant, la version entreprise de MongoDB ou un cluster dans MongoDB Atlas offre un outil supplémentaire pour automatiser le chiffrement des champs: *mongocryptd*.
Dans ce tutoriel, j'utiliserai le cryptage explicite (ou manuel) des champs qui ne nécessitent pas *mongocryptd* et l'édition entreprise de MongoDB ou Atlas. Si vous souhaitez explorer la version entreprise de CSFLE avec Java, vous pouvez en savoir plus dans cette documentation https://www.mongodb.com/docs/drivers/java/sync/v4.5/fundamentals/csfle/

Ne confondez pas mongocryptd avec la bibliothèque libmongocrypt qui est la bibliothèque C utilisée par les drivers pour chiffrer et déchiffrer vos données. Nous avons besoin de cette bibliothèque (en plus du driver Java) pour utiliser CSFLE. Je l'ai ajouté dans le fichier pom.xml de ce projet.

```
<dependency>
 <groupId>org.mongodb</groupId>
 <artifactId>mongodb-crypt</artifactId>
 <version>1.3.0</version>
</dependency>
```

Pour que les exemples de code soient courts et agréables dans les exemples ci-dessous, je ne partagerai que les parties les plus pertinentes. Si vous voulez voir le code fonctionner avec tout son contexte, veuillez télécharger le code source dans le dépôt github dans le package *csfle* directement.
https://github.com/mongodb-developer/java-quick-start/tree/master/src/main/java/com/mongodb/quickstart/csfle

Lancer le Code de Démarrage Rapide

Dans ce tutoriel, je vais vous montrer l'API CSFLE à l'aide du driver Java MongoDB. Je vais vous montrer comment :

- créer et configurer les connexions MongoDB dont nous avons besoin.
- créer une clé principale (Master Key en anglais).
- créer des clés de chiffrement des données (Data Encryption Key ou DEK en anglais).
- créer et lire des documents chiffrés.

Pour exécuter mon code à partir du dépôt Github ci-dessus, consultez le fichier README.md
https://github.com/mongodb-developer/java-quick-start/blob/master/README.md

Mais pour faire court, la commande suivante devrait vous permettre d'être opérationnel en un rien de temps :

```
mvn compile exec:java -Dexec.mainClass="com.mongodb.quickstart.csfle.ClientSideFieldLevelEncryption" -Dmongodb.uri="mongodb+srv://USERNAME:PASSWORD@cluster0-abcde.mongodb.net/test?w=majority"
```

Voici la sortie que vous devriez obtenir :

```
**************
* MASTER KEY *
**************
A new Master Key has been generated and saved to file "master_key.txt".
Master Key: [100, 82, 127, -61, -92, -93, 0, -11, 41, -96, 89, -39, -26, -25, -33, 37, 85, -50, 64, 70, -91, 99, -44, -57, 18, 105, -101, -111, -67, -81, -19, 56, -112, 62, 11, 106, -6, 85, -125, 49, -7, -49, 38, 81, 24, -48, -6, -15, 21, -120, -37, -5, 65, 82, 74, -84, -74, -65, -43, -15, 40, 80, -23, -52, -114, -18, -78, -64, -37, -3, -23, -33, 102, -44, 32, 65, 70, -123, -97, -49, -13, 126, 33, -63, -75, -52, 78, -5, -107, 91, 126, 103, 118, 104, 86, -79]

******************
* INITIALIZATION *
******************
=> Creating local Key Management System using the master key.
=> Creating encryption client.
=> Creating MongoDB client with automatic decryption.
=> Cleaning entire cluster.
```

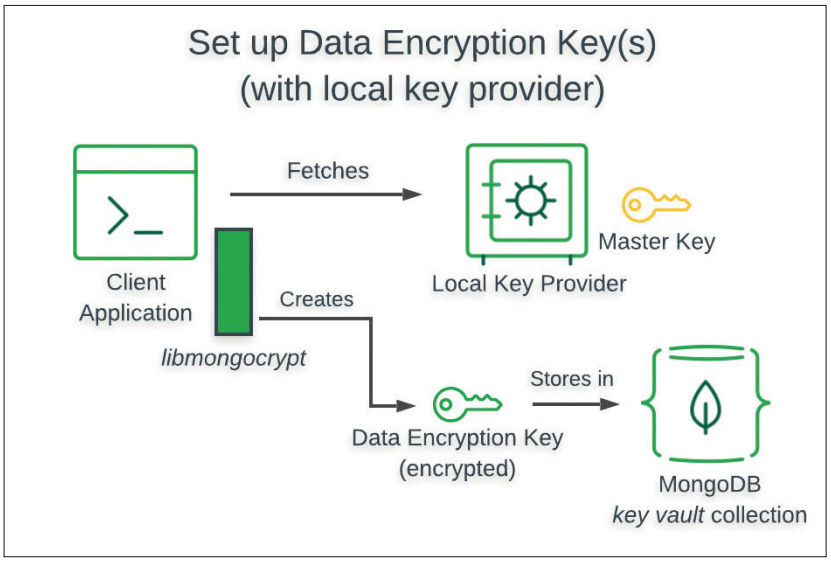


Figure 1

```
************************************
* CREATE KEY ALT NAMES UNIQUE INDEX *
************************************

*******************************
* CREATE DATA ENCRYPTION KEYS *
*******************************
Created Bobby's data key ID: 668a35af-df8f-4c41-9493-8d09d3d46d3b
Created Alice's data key ID: 003024b3-a3b6-490a-9f31-7abb7bcc334d

************************************************
* INSERT ENCRYPTED DOCUMENTS FOR BOBBY & ALICE *
************************************************
2 docs have been inserted.

*********************************
* FIND BOBBY'S DOCUMENT BY PHONE *
*********************************
Bobby document found by phone number:
{
  "_id": {
    "$oid": "60551bc8dd8b737958e3733f"
  },
  "name": "Bobby",
  "age": 33,
  "phone": "01 23 45 67 89",
  "blood_type": "A+",
  "medical_record": [
    {
      "test": "heart",
      "result": "bad"
    }
  ]
}

****************************
* READING ALICE'S DOCUMENT *
****************************
Before we remove Alice's key, we can read her document.
{
  "_id": {
    "$oid": "60551bc8dd8b737958e37340"
  },
  "name": "Alice",
  "age": 28,
  "phone": "09 87 65 43 21",
  "blood_type": "O+"
}

*****************************************************************
* REMOVE ALICE'S KEY + RESET THE CONNECTION (reset DEK cache) *
*****************************************************************
Alice key is now removed: 1 key removed.
=> Creating MongoDB client with automatic decryption.

****************************************
* TRY TO READ ALICE DOC AGAIN BUT FAIL *
****************************************
We get a MongoException because 'libmongocrypt' can't decrypt these fields anymore.
```

Regardons plus en profondeur pour comprendre ce qui se passe.

Comment ça marche **Figure 1**

CSFLE a l'air compliqué, comme toute fonctionnalité de sécurité et de chiffrement, je suppose.

Essayons de faire simple en quelques mots:

1. Nous avons besoin d'une clé principale qui déverrouille toutes les clés de chiffrement des données (DEK en anglais) que nous pouvons utiliser pour chiffrer un ou plusieurs champs dans nos documents.
2. Vous pouvez utiliser une DEK pour l'ensemble de votre cluster ou une DEK différente pour chaque champ de chaque document de votre cluster. C'est à vous de décider.
3. Les DEK sont stockées dans une collection dans un cluster MongoDB qui **ne doit pas nécessairement** être le même que celui qui contient les données chiffrées. Les DEK sont stockées **chiffrées**. Elles sont inutiles sans la clé principale qu'il faut protéger.
4. Vous pouvez utiliser le chiffrement manuel (édition communautaire) ou automatisé (enterprise advanced ou Atlas) des champs.
5. Le déchiffrement peut être manuel ou automatisé. Les deux font partie de l'édition communautaire de MongoDB. Dans cet article, j'utiliserai le cryptage manuel et le décryptage automatisé pour rester fidèle à l'édition communautaire de MongoDB.

Clé principale : https://www.mongodb.com/docs/drivers/security/client-side-field-level-encryption-guide/#std-label-fle-create-a-master-key

Clés de Chiffrement des données (DEK) : https://www.mongodb.com/docs/drivers/security/client-side-field-level-encryption-guide/#b.-create-a-data-encryption-key

Conformité RGPD

Les lois européennes imposent la protection des données et de la vie privée. Tout manquement peut entraîner des amendes non négligeables.

CSFLE est un excellent moyen d'économiser des millions de dollars/euros. https://en.wikipedia.org/wiki/GDPR_fines_and_notices

Par exemple, CSFLE pourrait être un excellent moyen d'appliquer la politique du "droit à l'oubli" https://gdpr-info.eu/art-17-gdpr/ de la RGPD. Si un utilisateur demande à être supprimé de vos systèmes, les données doivent être effacées de votre cluster de production, bien sûr, mais aussi des logs, de l'environnement de développement, et des sauvegardes... Et avouons-le nous : personne ne supprimera jamais les données des sauvegardes. Et si jamais vous restaurez ou utilisez ces sauvegardes, cela peut vous coûter des millions de dollars/euros.
Mais maintenant... chiffrer les données de chaque utilisateur avec une clé de chiffrement de données (DEK) unique et pour "oublier" un utilisateur pour toujours, il vous suffit de supprimer la clé. Ainsi, enregistrer les DEK sur un cluster séparé et appliquer une politique de faible rétention sur ce cluster garantira qu'un utilisateur est vraiment oublié pour toujours une fois la clé supprimée.
Kenneth White https://www.linkedin.com/in/biotech/, Security Principal chez MongoDB, qui a travaillé sur CSFLE, l'explique parfaitement dans cette réponse dans le Forum Communautaire de MongoDB.
https://developer.mongodb.com/community /forums/t/client-side-field-level-encryption-deks-and-backups/13577/2

Si la motivation principale est simplement de s'assurer que les enregistrements d'utilisateurs en clair qui ont été supprimés restent supprimés quoi qu'il arrive, alors cela devient une simple stratégie de synchronisation et de séparation des préoccupations, et la solution la plus simple consiste à déplacer la collection de coffres-forts (qui contient les DEK) vers une autre base de données ou cluster complètement séparé, configuré avec une rétention de sauvegarde beaucoup plus courte ; CSFLE ne suppose pas que votre collection de DEK chiffrés est co-résidente avec votre cluster actif ou a les mêmes contrôles d'accès et historique de sauvegarde, juste que le client peut, si nécessaire, établir une connexion authentifiée à cette base de données de coffres-forts. Il est important de noter cependant qu'avec un cycle de sauvegarde plus court, en cas de corruption de données catastrophique (malveillante, intentionnelle ou accidentelle), toutes les clés de cette base de données (et donc toutes les données chiffrées) ne sont récupérables qu'à partir de la sauvegarde la plus récente du coffre-fort.

Plus trivial, mais en cas d'intrusion, toute donnée volée sera sans valeur sans la clé principale et n'entraînera pas une amende ruineuse.

La Clé Maître

La clé principale est un tableau de 96 octets. Il peut être stocké dans un service de gestion de clés dans un fournisseur de cloud ou peut être géré localement.
https://www.mongodb.com/docs/drivers/security/client-side-field-level-encryption-local-key- à-kms/

D'une manière ou d'une autre, vous devez la protéger de toute menace.
C'est aussi simple que cela de générer une nouvelle clé principale :

```
final byte[] masterKey = new byte[96];
new SecureRandom().nextBytes(masterKey);
```

Mais vous voudrez probablement le faire une fois, puis réutiliser la même à chaque redémarrage de votre application.
Voici mon implémentation pour la stocker dans un fichier local la première fois puis la réutiliser à chaque redémarrage.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.SecureRandom;
import java.util.Arrays;

public class MasterKey {

    private static final int SIZE_MASTER_KEY = 96;
    private static final String MASTER_KEY_FILENAME = "master_key.txt";

    public static void main(String[] args) {
        new MasterKey().tutorial();
    }

    private void tutorial() {
        final byte[] masterKey = generateNewOrRetrieveMasterKeyFrom
File(MASTER_KEY_FILENAME);
        System.out.println("Master Key: " + Arrays.toString(masterKey));
    }

    private byte[] generateNewOrRetrieveMasterKeyFromFile(String
filename) {
        byte[] masterKey = new byte[SIZE_MASTER_KEY];
        try {
            retrieveMasterKeyFromFile(filename, masterKey);
            System.out.println("An existing Master Key was found in file \""
+ filename + "\".");
        } catch (IOException e) {
            masterKey = generateMasterKey();
            saveMasterKeyToFile(filename, masterKey);
            System.out.println("A new Master Key has been generated and
 saved to file \"" + filename + "\".");
        }
        return masterKey;
    }

    private void retrieveMasterKeyFromFile(String filename, byte[]
masterKey) throws IOException {
        try (FileInputStream fis = new FileInputStream(filename)) {
            fis.read(masterKey, 0, SIZE_MASTER_KEY);
        }
    }

    private byte[] generateMasterKey() {
        byte[] masterKey = new byte[SIZE_MASTER_KEY];
        new SecureRandom().nextBytes(masterKey);
        return masterKey;
    }

    private void saveMasterKeyToFile(String filename, byte[] masterKey) {
        try (FileOutputStream fos = new FileOutputStream(filename)) {
            fos.write(masterKey);
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

C'est loin d'être sûr pour un environnement de production, car laisser le fichier *master_key.txt* directement dans le dossier de l'application sur votre serveur de production revient à laisser la combinaison de coffre-fort sur un post-it. Sécurisez ce fichier ou envisagez d'utiliser un KMS en production.
https://www.mongodb.com/docs/drivers/security/client-side-field-level-encryption-local-key-to-kms/

Dans ce tutoriel, je n'utiliserai qu'une seule clé principale, mais il est tout à fait possible d'utiliser plusieurs clés principales.

Le Fournisseur de Services de Gestion de Clés (KMS)

Quelle que soit la solution que vous choisissez pour la clé principale, vous aurez besoin d'un fournisseur KMS pour configurer les *ClientEncryptionSettings* et les *AutoEncryption Settings*.
https://www.mongodb.com/docs/drivers/security/client-side-field-level-encryption-local-key-to-kms

Voici la configuration pour un KMS local :

```
Map<String, Map<String, Object>> kmsProviders = new HashMap
<String, Map<String, Object>>() {{
    put("local", new HashMap<String, Object>() {{
        put("key", localMasterKey);
    }});
}};
```

Les Clients

Nous devrons configurer deux clients différents :

- Le premier *ClientEncryption* sera utilisé pour créer nos Data Encryption Keys (DEK) et chiffrer nos champs manuellement.
- Le second *MongoClient* sera la connexion MongoDB plus conventionnelle que nous utiliserons pour lire et écrire nos documents, à la différence qu'elle sera configurée pour décrypter automatiquement les champs chiffrés.

ClientEncryption

```
ConnectionString connection_string = new ConnectionString("mongodb
://localhost");
MongoClientSettings kvmcs = MongoClientSettings.builder().apply
ConnectionString(connection_string).build();

ClientEncryptionSettings ces = ClientEncryptionSettings.builder()
                                .keyVaultMongoClientSettings(kvmcs)
                                .keyVaultNamespace("csfle.vault")
                                .kmsProviders(kmsProviders)
                                .build();

ClientEncryption encryption = ClientEncryptions.create(ces);
```

MongoClient

```
AutoEncryptionSettings aes = AutoEncryptionSettings.builder()
                                .keyVaultNamespace("csfle.vault")
                                .kmsProviders(kmsProviders)
                                .bypassAutoEncryption(true)
                                .build();

MongoClientSettings mcs = MongoClientSettings.builder()
                            .applyConnectionString(connection_string)
                            .autoEncryptionSettings(aes)
                            .build();

MongoClient client = MongoClients.create(mcs);
```

bypassAutoEncryption(true) est le ticket pour l'édition communautaire. Sans cela, mongocryptd s'appuierait sur le schéma JSON que vous auriez à fournir pour crypter automatiquement les documents. Voir cet exemple dans la documentation pour plus d'informations : https://mongodb.github.io/mongo-java-driver/4.2/driver/tutorials/client-side-encryption/#examples.

Vous n'êtes pas obligé de réutiliser la même URI de connexion pour les deux connexions à MongoDB. Il serait en fait beaucoup plus "compatible avec la RGPD" d'utiliser des clusters séparés afin que vous puissiez appliquer une politique de faible rétention sur les clés de chiffrement des données.

Index Unique sur les Noms Alternatifs des Clés

La première chose à faire avant de créer votre première clé de chiffrement de données est de créer un index unique sur les noms alternatifs des clés pour vous assurer que vous ne pouvez pas réutiliser le même nom alternatif sur deux DEK différentes. Ces noms vous aideront à "étiqueter" vos clés pour savoir à quoi chacune sert ce qui est totalement à vous de décider.

```
MongoCollection<Document> vaultColl = client.getDatabase("csfle")
.getCollection("vault");
vaultColl.createIndex(ascending("keyAltNames"),
            new IndexOptions().unique(true).partialFilterExpression
(exists("keyAltNames")));
```

Dans mon exemple, je choisis d'utiliser une DEK par utilisateur. Je chiffrerai tous les champs que je souhaite sécuriser dans chaque document utilisateur avec la même clé. Si je veux "oublier" un utilisateur, il me suffit de supprimer cette clé. Dans mon exemple, les noms sont uniques, donc j'utilise ceci pour mes *keyAltNames*. C'est un excellent moyen d'être en conformité avec la RGPD.

Créer des Clés de Chiffrement de Données

Créons deux clés de chiffrement de données : une pour Bobby et une pour Alice. Chacune sera utilisée pour chiffrer tous les champs que je souhaite garder en sécurité dans mes documents pour chaque utilisateur.

```
BsonBinary bobbyKeyId = encryption.createDataKey("local", keyAlt
Name("Bobby"));
BsonBinary aliceKeyId = encryption.createDataKey("local", keyAlt
Name("Alice"));
```

Nous recevons un peu d'aide de cette méthode privée pour rendre mon code plus facile à lire :

```
private DataKeyOptions keyAltName(String altName) {
  return new DataKeyOptions().keyAltNames(singletonList(altName));
}
```

Voici à quoi ressemble le DEK de Bobby dans ma collection *csfle.vault* :

```
{
 "_id" : UUID("aaa2e53d-875e-49d8-9ce0-dec9a9658571"),
 "keyAltNames" : [ "Bobby" ],
 "keyMaterial" : BinData(0,"/ozPZBMNUJU9udZyTYe1hX/KHqJJPrjd
Pads8UNjHX+cZVklXnweZe5pGPpzcVcGmYctTAdxB3b+ImY5ONTzEZkq
Mg8JIWenIWQVY5fogIpfHDJQyIQoEjXV3+e3ZY1WmWJR8mOp7pMoTy
oGlZU2TwyqT9fcN7E5pNRh0uL3kCPk0s0OxLT/ejQISoY/wxq2uvyIK/C6/L
rD1ymIC9w6YA=="),
 "creationDate" : ISODate("2021-03-19T16:16:09.800Z"),
 "updateDate" : ISODate("2021-03-19T16:16:09.800Z"),
 "status" : 0,
 "masterKey" : {
  "provider" : "local"
 }
}
```

Comme vous pouvez le voir ci-dessus, le *keyMaterial* (la DEK elle-même) est chiffré par la clé principale. Sans la clé principale pour la déchiffrer, elle est inutile. De plus, vous pouvez identifier qu'il s'agit de la clé de Bobby dans le champ *keyAltNames*.

Créer des Documents Chiffrés

Maintenant que nous avons une clé de chiffrement pour Bobby et Alice, je peux créer leurs documents respectifs et les insérer dans MongoDB comme suit :

```
private static final String DETERMINISTIC = "AEAD_AES_256_CBC_
HMAC_SHA_512-Deterministic";
private static final String RANDOM = "AEAD_AES_256_CBC_HMAC_
SHA_512-Random";

private Document createBobbyDoc(ClientEncryption encryption) {
  BsonBinary phone = encryption.encrypt(new BsonString("01 23 45
 67 89"), deterministic(BOBBY));
  BsonBinary bloodType = encryption.encrypt(new BsonString("A+"),
 random(BOBBY));
  BsonDocument medicalEntry = new BsonDocument("test", new Bson
String("heart")).append("result", new BsonString("bad"));
  BsonBinary medicalRecord = encryption.encrypt(new BsonArray
(singletonList(medicalEntry)), random(BOBBY));
  return new Document("name", BOBBY).append("age", 33)
                    .append("phone", phone)
                    .append("blood_type", bloodType)
                    .append("medical_record", medicalRecord);
}

private Document createAliceDoc(ClientEncryption encryption) {
  BsonBinary phone = encryption.encrypt(new BsonString("09 87 65
 43 21"), deterministic(ALICE));
  BsonBinary bloodType = encryption.encrypt(new BsonString("O+"),
random(ALICE));
  return new Document("name", ALICE).append("age", 28).append
("phone", phone).append("blood_type", bloodType);
}

private EncryptOptions deterministic(String keyAltName) {
 return new EncryptOptions(DETERMINISTIC).keyAltName(keyAltName);
}

private EncryptOptions random(String keyAltName) {
  return new EncryptOptions(RANDOM).keyAltName(keyAltName);
}

private void createAndInsertBobbyAndAlice(ClientEncryption encryption,
 MongoCollection<Document> usersColl) {
  Document bobby = createBobbyDoc(encryption);
  Document alice = createAliceDoc(encryption);
 int nbInsertedDocs = usersColl.insertMany(asList(bobby, alice)).get
InsertedIds().size();
  System.out.println(nbInsertedDocs + " docs have been inserted.");
}
```

Voici à quoi ressemblent les documents Bobby et Alice dans ma collection *encrypted.users* :

Bobby

```
{
 "_id" : ObjectId("6054d91c26a275034fe53300"),
 "name" : "Bobby",
 "age" : 33,
 "phone" : BinData(6,"ATKkRdZWR0+HpqNyYA7zgIUCgeBE4SvLRwa
Xz/rFI8NPZsirWdHRE51pPa/2W9xgZ13InHd56J1PLu9uv/hSkBgajE+MJL
wQvJUkXatOJGbZd56BizxyKKTH+iy+8vV7CmY="),
 "blood_type" : BinData(6,"AjKkRdZWR0+HpqNyYA7zgIUCUdc30A8I
Ti2i1pWn7CRpz60yrDps7A8gUJhJdj+BEqIlx9xSUQ7xpnc/6ri2/+ostFtxIq
/b6IQArGi+8ZBISw=="),
 "medical_record" : BinData(6,"AjKkRdZWR0+HpqNyYA7zgIUESl5s4t
PPvzqwe788XF8o91+JNqOUgo5kiZDKZ8qudloPutr6S5cE8iHAJ0AsbZDYq
7XCqbqiXvjQobObvsIR90xJvVMQidHzWtqWMlzig6ejdZQswz2/WT78Rr0
N8awO")
}
```

Alice

```
{
 "_id" : ObjectId("6054d91c26a275034fe53301"),
 "name" : "Alice",
 "age" : 28,
 "phone" : BinData(6,"AX7Xd65LHUcWgYj+KbUT++sCC6xaCZ1zaM
tzabawAgB79quwKvld8fpA+0m+CtGevGyIgVRjtj2jAHAOvREsoy3oq9p5
mbJvnBqi8NttHUJpqooUn22Wx7o+nlo633QO8+c="),
 "blood_type" : BinData(6,"An7Xd65LHUcWgYj+KbUT++sCTyp+PJ
XudAKM5HcdX21vB0VBHqEXYSplHdZR0sCOxzBMPanVsTRrOSdAK5yHTh
P3Vitsu9jlbNo+lz5f3L7KYQ==")
}
```

Le chiffrement des champs côté client fournit actuellement deux algorithmes différents pour chiffrer les données que vous souhaitez sécuriser.
https://www.mongodb.com/docs/manual/core/security-client-side-encryption/ #encryption-algorithms

AEAD_AES_256_CBC_HMAC_SHA_512-Deterministic

Avec cet algorithme, le résultat du chiffrement avec les mêmes données en entrée (valeur et DEK) est déterministe. Cela signifie que nous avons un meilleur support pour les opérations de lecture, mais les données chiffrées avec une faible cardinalité sont sensibles aux attaques par analyse de fréquence.

https://en.wikipedia.org/wiki/Deterministic_encryption
https://en.wikipedia.org/wiki/Cardinality
https://en.wikipedia.org/wiki/Frequency_analysis

Dans mon exemple, si je veux pouvoir récupérer les utilisateurs par numéros de téléphone, je dois utiliser l'algorithme déterministe. Comme un numéro de téléphone est susceptible d'être unique dans ma collection d'utilisateurs, cet algorithme est sûr ici pour protéger ce champ.

AEAD_AES_256_CBC_HMAC_SHA_512-Random

Avec cet algorithme, le résultat du cryptage est *toujours* différent. Cela signifie qu'il offre les meilleures garanties de confidentialité des données, même lorsque la cardinalité est faible, mais empêche les opérations de lecture basées sur ces champs. https://en.wikipedia.org/wiki/Probabilistic_encryption

Dans mon exemple, le groupe sanguin a une faible cardinalité et cela n'a aucun sens de rechercher dans ma collection d'utilisateurs par groupe sanguin de toute façon. Il est donc sûr d'utiliser cet algorithme pour ce champ.

De plus, le dossier médical de Bobby doit être très sécurisé. Ainsi, l'ensemble du sous-document contenant tous ses dossiers médicaux est également chiffré avec l'algorithme aléatoire et ne sera de toute façon pas utilisé pour rechercher Bobby dans ma collection.

Lire le Document de Bobby

Comme mentionné dans la section précédente, il est possible de rechercher des documents par champs chiffrés avec l'algorithme déterministe.

Voici comment:

```
BsonBinary phone = encryption.encrypt(new BsonString("01 23 45 67 89"), deterministic(BOBBY));
String doc = usersColl.find(eq("phone", phone)).first().toJson();
```

Je chiffre simplement à nouveau, avec la même clé, le numéro de téléphone que je cherche, et je peux utiliser ce *BsonBinary* dans ma requête pour trouver Bobby.

Voici le résultat que j'obtiens si j'affiche le document en chaîne de caractères :

```
{
 "_id": {
  "$oid": "6054d91c26a275034fe53300"
 },
 "name": "Bobby",
 "age": 33,
 "phone": "01 23 45 67 89",
 "blood_type": "A+",
 "medical_record": [
  {
   "test": "heart",
   "result": "bad"
  }
 ]
}
```

Comme vous pouvez le voir, le déchiffrage automatique a fonctionné comme prévu, je peux voir mon document en texte clair. Pour trouver ce document, je pourrais utiliser les champs *_id*, *name*, *age*, ou *phone*, mais pas les champs *blood_type* ou *medical_record*.

Lire le Document d'Alice

Mettons maintenant CSFLE à l'épreuve. Je veux être sûr que si la DEK d'Alice est détruite, le document d'Alice est perdu à jamais et ne pourra jamais être restauré, même à partir d'une sauvegarde qui pourrait être restaurée. C'est pourquoi il est important de conserver les DEK et les documents chiffrés dans deux clusters différents qui n'ont pas la même politique de rétention des sauvegardes.

Récupérons le document d'Alice par son nom, mais protégeons mon code au cas où la clé d'Alice soit "perdue par inadvertance"...

```
private void readAliceIfPossible(MongoCollection<Document> usersColl) {
    try {
     String aliceDoc = usersColl.find(eq("name", ALICE)).first().toJson();
     System.out.println("Before we remove Alice's key, we can read her document.");
      System.out.println(aliceDoc);
    } catch (MongoException e) {
     System.err.println("We get a MongoException because 'libmongocrypt' can't decrypt these fields anymore.");
    }
}
```

Si sa clé existe toujours dans la base de données, je peux alors déchiffrer son document :

```
{
 "_id": {
  "$oid": "6054d91c26a275034fe53301"
 },
 "name": "Alice",
 "age": 28,
 "phone": "09 87 65 43 21",
 "blood_type": "O+"
}
```

Maintenant, supprimons sa clé de la base de données :

```
vaultColl.deleteOne(eq("keyAltNames", ALICE));
```

Dans un vrai environnement de production, cela n'aurait aucun sens de relire son document ; et parce que nous sommes tous des développeurs professionnels et organisés qui aiment garder les choses en ordre, nous supprimerons également le document d'Alice avec sa DEK, car ce document est maintenant complètement sans valeur pour nous de toute façon.

Dans mon exemple, je veux quand même essayer de lire ce document. Mais si j'essaie de le lire immédiatement après avoir supprimé son document, il y a de grandes chances que je puisse toujours le faire à cause du cache de clé de chiffrement de données de 60 secondes qui est géré par *libmongocrypt*.
https://github.com/mongodb/specifications/blob/master/source/client-side-encryption/client-side-encryption.rst#libmongocrypt-data-key-caching

Ce cache est très important car, sans lui, de multiples allers-retours seraient nécessaires pour déchiffrer mes documents. Il est essentiel pour empêcher CSFLE d'impacter les performances de votre cluster MongoDB.
Donc, pour m'assurer que je n'utilise plus ce cache, je crée un tout nouveau *MongoClient* (toujours avec les paramètres de déchiffrement automatique) pour cet exemple. Mais bien sûr, en production, cela n'aurait aucun sens de le faire.
Maintenant, si j'essaye d'accéder à nouveau au document d'Alice, j'obtiens la *MongoException* suivante, comme prévu et le document d'Alice est bien perdu à tout jamais :

```
com.mongodb.MongoException: nocom.mongodb.MongoException: not all keys requested were satisfied
    at com.mongodb.MongoException.fromThrowableNonNull(MongoException.java:83)
    at com.mongodb.client.internal.Crypt.fetchKeys(Crypt.java:286)
    at com.mongodb.client.internal.Crypt.executeStateMachine(Crypt.java:244)
    at com.mongodb.client.internal.Crypt.decrypt(Crypt.java:128)
    at com.mongodb.client.internal.CryptConnection.command(CryptConnection.java:121)
    at com.mongodb.client.internal.CryptConnection.command(CryptConnection.java:131)
    at com.mongodb.internal.operation.CommandOperationHelper.executeCommand(CommandOperationHelper.java:345)
    at com.mongodb.internal.operation.CommandOperationHelper.executeCommand(CommandOperationHelper.java:336)
    at com.mongodb.internal.operation.CommandOperationHelper.executeCommandWithConnection(CommandOperationHelper.java:222)
    at com.mongodb.internal.operation.FindOperation$1.call(FindOperation.java:658)
    at com.mongodb.internal.operation.FindOperation$1.call(FindOperation.java:652)
    at com.mongodb.internal.operation.OperationHelper.withReadConnectionSource(OperationHelper.java:583)
    at com.mongodb.internal.operation.FindOperation.execute(FindOperation.java:652)
    at com.mongodb.internal.operation.FindOperation.execute(FindOperation.java:80)
    at com.mongodb.client.internal.MongoClientDelegate$DelegateOperationExecutor.execute(MongoClientDelegate.java:170)
    at com.mongodb.client.internal.FindIterableImpl.first(FindIterableImpl.java:200)
    at com.mongodb.quickstart.csfle.ClientSideFieldLevelEncryption.readAliceIfPossible(ClientSideFieldLevelEncryption.java:91)
    at com.mongodb.quickstart.csfle.ClientSideFieldLevelEncryption.demo(ClientSideFieldLevelEncryption.java:79)
    at com.mongodb.quickstart.csfle.ClientSideFieldLevelEncryption.main(ClientSideFieldLevelEncryption.java:41)
    Caused by: com.mongodb.crypt.capi.MongoCryptException: not all keys requested were satisfied
    at com.mongodb.crypt.capi.MongoCryptContextImpl.throwExceptionFromStatus(MongoCryptContextImpl.java:145)
    at com.mongodb.crypt.capi.MongoCryptContextImpl.throwExceptionFromStatus(MongoCryptContextImpl.java:151)
    at com.mongodb.crypt.capi.MongoCryptContextImpl.completeMongoOperation(MongoCryptContextImpl.java:93)
    at com.mongodb.client.internal.Crypt.fetchKeys(Crypt.java:284)
    ... 17 more
```

Conclusion

Dans cet article, nous avons découvert comment utiliser le chiffrement des champs côté client à l'aide du driver Java MongoDB, en utilisant uniquement l'édition communautaire de MongoDB. Vous pouvez en savoir plus sur le chiffrement automatisé dans la documentation.
https://www.mongodb.com/docs/manual/core/security-automatic-client-side-encryption
CSFLE est la fonctionnalité de sécurité ultime pour assurer le niveau de sécurité maximal de votre cluster. Même vos administrateurs ne pourront pas accéder aux données en production s'ils n'ont pas accès aux clés principales.
Mais ce n'est pas la seule mesure de sécurité que vous devez utiliser pour protéger votre cluster. Empêcher l'accès à votre cluster est, bien sûr, la première mesure de sécurité que vous devez appliquer en activant l'authentification et en limitant l'exposition du réseau.
https://www.mongodb.com/docs/manual/tutorial/enable-authentication/
https://www.mongodb.com/docs/manual/administration/security-checklist/#arrow-limit-network-exposure
En cas de doute, consultez la liste des contrôles de sécurité avant de lancer un cluster en production pour vous assurer que vous n'avez négligé aucune option de sécurité que MongoDB a à offrir pour protéger vos données.
https://www.mongodb.com/docs/manual/administration/security-checklist/

Il y a beaucoup de flexibilité dans la mise en œuvre de CSFLE : vous pouvez choisir d'utiliser une ou plusieurs clés principales, idem pour les clés de chiffrement des données. Vous pouvez également choisir de chiffrer tous vos numéros de téléphone dans votre collection avec le même DEK ou d'en utiliser une différente pour chaque utilisateur. C'est vraiment à vous de décider comment vous organisez votre stratégie de chiffrement mais, bien sûr, assurez-vous qu'elle remplit bien toutes vos obligations légales. Il existe plusieurs bonnes manières de mettre en œuvre CSFLE, alors assurez-vous de trouver la plus adaptée à votre cas d'utilisation.

Documentations supplémentaires

Dépôt GitHub avec tous les exemples Java Quickstart de cette série d'article :
https://github.com/mongodb-developer/java-quick-start
MongoDB CSFLE Doc:
https://www.mongodb.com/docs/manual/core/security-client-side-encryption
MongoDB Java Driver CSFLE Doc :
https://www.mongodb.com/docs/drivers/security/client-side-field-level-encryption-guide/
Exemple d'implémentation CSFLE de l'Université MongoDB :
https://github.com/mongodb-university/csfle-guides/tree/master/java/

Carvel : une boîte à outils pour vos apps Kubernetes

Carvel (anciennement k14s) est une suite d'outils open source destinée à simplifier la configuration, le packaging et le déploiement d'applications sur Kubernetes (K8s). Contrairement à des applications plus polyvalentes, comme Helm, qui adoptent une approche tout-en-un, Carvel est composée d'outils simples, mais intégrés, qui répondent chacun à une mission unique et interagissent facilement les uns avec les autres.

Alexandre CAUSSIGNAC

Solution Engineer Manager chez VMware

Ma mission est de conseiller et d'accompagner les entreprises dans leur transformation digitale. Au-delà des technologies, j'adore échanger avec les gens sur leurs besoins finaux, car je pense que les seules barrières à ce jour dans notre secteur d'activité sont notre imagination et non notre savoir-faire technique.

Élise SAVORNIN

Academy Digital Solution Engineer, EUC chez VMware

Tout juste diplômée de L'ISEP - école d'ingénieur du numérique, je travaille avec mon équipe pour aider les entreprises à perfectionner leur environnement utilisateur afin de leur apporter la meilleure expérience de travail possible. Ayant tout juste rejoint l'équipe, je suis très curieuse du monde technologique qui nous entoure et adore découvrir de nouveaux outils.

Cette approche "Do One Thing And Do It Well", fidèle à la philosophie Unix, facilite son utilisation au quotidien, sa prise en main, son débogage ainsi que son intégration dans un écosystème plus large (par exemple, des chaînes de CI/CD). À ce jour la suite Carvel compte six outils. D'autres sont en cours de développement(1).

Le but de cet article est de présenter ces outils, le besoin auquel ils répondent et la valeur qu'ils peuvent apporter au processus de création d'une application ainsi qu'à sa maintenance (Day 1 & Day 2)

Le développement et la maintenance d'applications sur K8s se décomposent en plusieurs étapes :

0. La création de code source
1. La création de fichiers de configuration
2. La construction d'images
3. Le packaging
4. Le déploiement
5. La gestion du cycle de vie

Figure 1

Installation

Vous trouverez la procédure d'installation sur le site officiel de Carvel : https://carvel.dev/ (2). Elle ne sera pas décrite dans cet article, mais sachez qu'elle est très simple. Les outils qui composent la suite étant largement indépendants, vous pouvez choisir de ne pas tous les installer. Pour tester ces outils, vous trouverez un workshop sur le Tanzu Developer Center : https://tanzu.vmware.com/developer/workshops/lab-getting-started-with-carvel/(3)

Présentation des outils

vendir : synchronisation de dépendances externes

vendir répond à la nécessité de ne pas dépendre de dépendances externes, en automatisant leur synchronisation. Grâce à vendir, il est facile de consolider des ressources issues d'autres projets au sein d'un même référentiel.

Figure 2, un exemple de projet contenant une dépendance vers un fichier ***manifests/vanilla/vsphere-csi-driver.yaml***. Ce fichier sera copié dans les fichiers de configuration du projet suivant le chemin ***config/upstream/vsphere-csi***. *(4)*

Figure 3, le fichier est bien présent. À chaque nouvelle synchronisation, ce fichier sera mis à jour à partir de la source définie dans le YAML (exemple **Figure 2**). L'utilisateur ne doit donc pas le modifier. *(5)*

Couplé à ytt, il est simple de récupérer des ressources externes et de les modifier automatiquement sans avoir à faire de configuration « manuelle » autre que la configuration initiale.

ytt : templating de fichiers YAML

Le YAML est un langage commun dans l'écosystème Kubernetes, utilisé pour écrire des fichiers de configurations (applicatifs comme pour les clusters). Avec la multiplication du nombre de fichiers de configurations il devient vite difficile de standardiser, d'industrialiser et de limiter l'erreur humaine. Les outils qui existent pour résoudre ce problème sont Helm et Kustomize. Le projet Carvel possède également son propre outil appelé ytt, qui prend en charge les deux approches.

L'approche de ytt pour travailler avec ces fichiers YAML est la suivante. Au lieu d'interpréter la configuration comme du

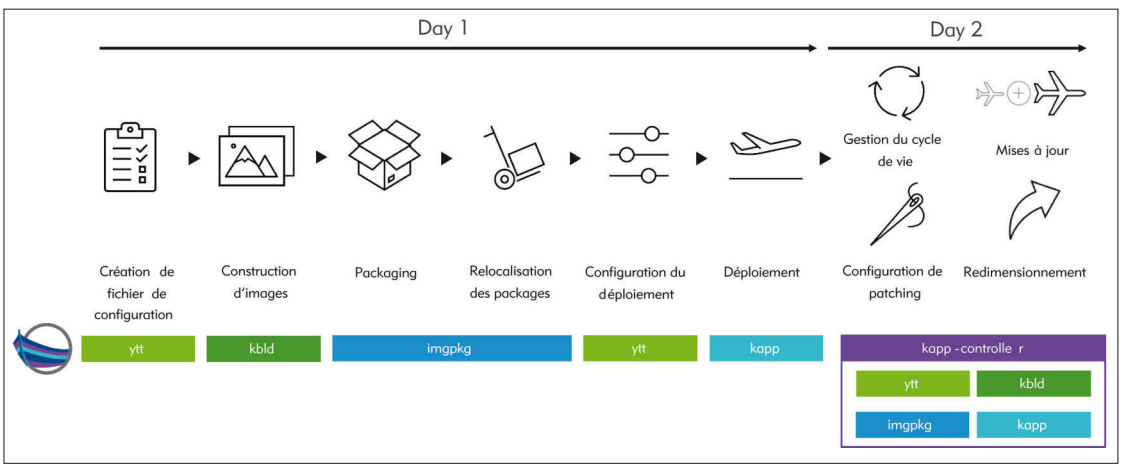


Figure 1 - *Schéma des outils inclus dans Carvel intégré à un cycle Day 1 et Day 2*

texte brut, il fonctionne avec des structures telles que des cartes, des listes, des documents, des scalaires, etc. Ce faisant, ytt est en mesure d'éliminer de nombreux problèmes tels que l'échappement de caractères, les ambiguïtés, etc. Voici une illustration de l'ensemble des structures gérées par ytt : **Figure 4** – (6)

ytt met à disposition un langage de type Python (Starlark) qui rend son utilisation très accessible.(7)

Grâce à la structure de ytt, il est possible de « variabiliser » certains paramètres de notre YAML afin de pouvoir modifier l'affichage sans avoir à modifier le code source de l'application. Dans l'exemple **Figure 5**, l'encadré vert indique à ytt que la valeur de la variable d'environnement ***HELLO_MSG*** sera définie dans le ***data.value.hello_msg***. La structure de ***data.values*** vient de la librairie ***data*** de ytt permettant de créer des options de configuration dans des fichiers séparés.

Ainsi, il est possible de modifier la valeur de cette variable, sans modifier le code source de l'application ni les fichiers de configuration. Voici un exemple de commande pour modifier la valeur de *HELLO_MSG*. **Figure 6**

Pour tester les fonctionnalités de ytt, vous pouvez vous rendre sur « le terrain de jeu interactif » de Carvel : https://carvel.dev/ytt/#playground.

Une des autres fonctionnalités intéressantes de ytt est l'overlay, qui permet de patcher des structures de fichier YAML déjà existantes. On peut l'utiliser pour rechercher, insérer, remplacer, ou supprimer des éléments de ces fichiers. Voici un exemple de commande pour remplacer le nombre de replicas par 3 pour tous les fichiers YAML concernés.

Figure 7 – (8)

kbld : construction d'images

Pour reconstruire une image de conteneur lors du changement du code source de l'application, il faut généralement exécuter manuellement un ***build*** à l'aide d'outils de création d'image tels que docker, pack, etc. ; puis envoyer l'image résultante dans un registre d'images accessible au cluster Kubernetes afin qu'elle puisse être déployée. Toute ressource de déploiement devra alors être modifiée pour utiliser la version spécifique de la nouvelle image.

L'outil kbld de Carvel est un outil qui fournit un moyen simple d'insérer la création d'images de conteneur dans un processus de déploiement. kbld recherche des images dans la configuration de l'application (plus précisément des clés d'image), vérifie s'il existe une définition de code source associée et, si c'est le cas, déclenche une construction de l'image du conteneur à l'aide de docker (ou d'un autre mécanisme de construction), et enfin capture la signature unique des images construites (sha256) et met à jour la configuration avec les nouvelles références d'image.

kbld peut s'intégrer entre ytt et kapp afin de déclencher une génération d'image répondant à la configuration reçue de ytt et en incluant le résumé de l'image avant de transmettre la configuration à kapp pour le déploiement. **Figure 8**, voici

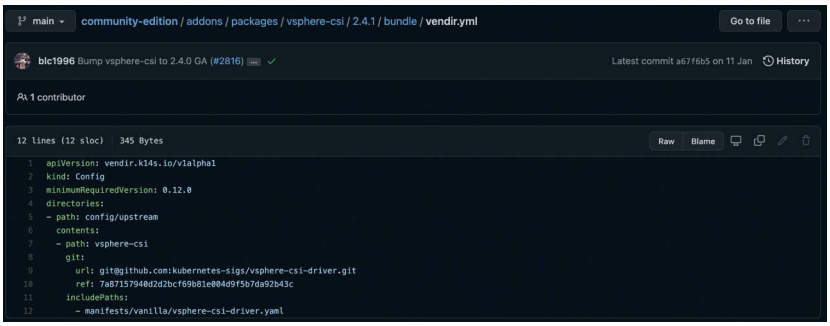


Figure 2 – *Exemple d'un projet de Tanzu Community*

Figure 3 – *Suite de l'exemple*

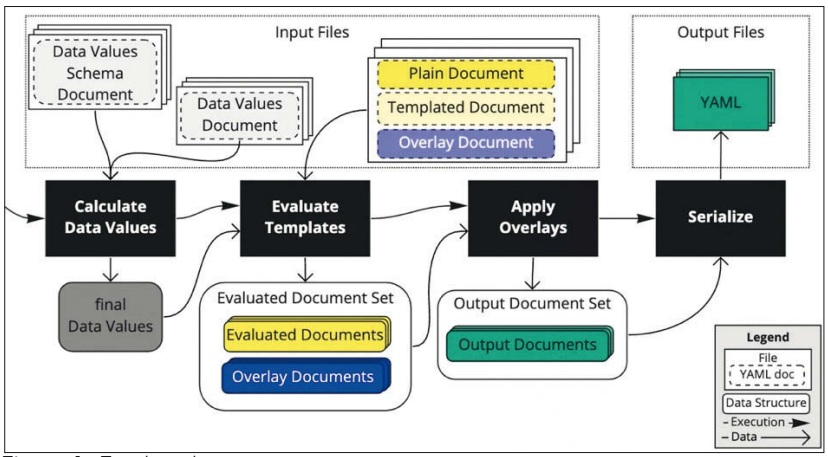


Figure 4 – *Type de pipeline*

```
[~/exercises] $ cat config-step-2-template/config.yml
#@ load("@ytt:data", "data")

#@ def labels():
simple-app: ""
#@ end

---
apiVersion: v1
kind: Service
metadata:
  name: simple-app
spec:
  ports:
  - port: #@ data.values.svc_port
    targetPort: #@ data.values.app_port
  selector: #@ labels()
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: simple-app
spec:
  selector:
    matchLabels: #@ labels()
  template:
    metadata:
      labels: #@ labels()
    spec:
      containers:
      - name: simple-app
        image: quay.io/eduk8s-labs/sample-app-go@sha256:5021a23e0c4a4633bfd6c95b13898cffb88a0e67f109d87ec01b4f896f4b4296
        env:
        - name: HELLO_MSG
          value: #@ data.values.hello_msg
```

```
#@data/values
---
svc_port: 8080
app_port: 8080
hello_msg: stranger
```

Figure 5 – *Exemple d'un YAML de configuration avec le fichier data associé*

```
ytt template -f config-step-2-template/ -v hello_msg="carvel user" | kapp deploy -a simple-app -f- --diff-changes --yes
```

Figure 6 – *Exemple d'une commande permettant de changer la valeur d'une variable d'environnement*

```
#@ load("@ytt:overlay", "overlay")
#@overlay/match by=overlay.subset({"kind": "Deployment", "metadata": {"name": "nginx-de
ployment"}})
---
spec:
  #! change spec.replicas to 3
  replicas: 3
```

Figure 7 – *Exemple issu du blog de Stéphane Robert. N'hésitez pas à consulter son super travail de vulgarisation de la techno.*

```
ytt template -f config-step-3-build-local/ -v hello_msg="carvel user" | kbld -f- | kapp deploy -a simple-
app -f- --diff-changes --yes
```

Figure 8 – *Exemple d'une ligne de commande utilisant ytt, kbld et kapp*

```
$ imgpkg copy -b index.docker.io/user1/simple-app-bundle:v1.0.0 --to-repo registry.corp.com/apps/simple-app-bundle

copy | exporting 2 images...
copy | will export index.docker.io/user1/simple-app-bundle@sha256:4c8b96d4fffdfae29258d94a22ae4ad1fe36139d47288b8960d9958d1e63a9d0
copy | will export index.docker.io/user1/simple-app-bundle@sha256:70225df0a05137ac385c95eb69f89ded3e7ef3a0c34db43d7274fd9eba3705bb
copy | exported 2 images
copy | importing 2 images...
copy | importing index.docker.io/user1/simple-app-bundle@sha256:70225df0a05137ac385c95eb69f89ded3e7ef3a0c34db43d7274fd9eba3705bb
       -> registry.corp.com/apps/simple-app-bundle@sha256:70225df0a05137ac385c95eb69f89ded3e7ef3a0c34db43d7274fd9eba3705bb...
copy | importing index.docker.io/user1/simple-app-bundle@sha256:4c8b96d4fffdfae29258d94a22ae4ad1fe36139d47288b8960d9958d1e63a9d0
       -> registry.corp.com/apps/simple-app-bundle@sha256:4c8b96d4fffdfae29258d94a22ae4ad1fe36139d47288b8960d9958d1e63a9d0...
copy | imported 2 images
Succeeded
```

Figure 9 - *Exemple d'une commande imgpkg qui déplace une image entre deux registries connectées*

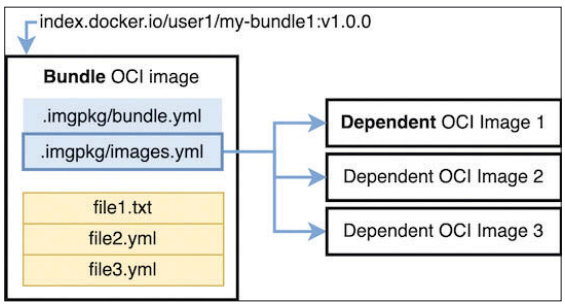


Figure 10 – *Schéma du concept d'un bundle avec imgpkg*

```
[~/exercises] $ kapp inspect -a simple-app --tree
Target cluster 'https://10.96.0.1:443'

Resources in app 'simple-app'

Namespace            Name                              Kind        Owner    Conds.  Rs  Ri  Age
eduk8s-labs-w02-s045  simple-app                       Deployment  kapp     2/2 t   ok  -   20s
eduk8s-labs-w02-s045   L simple-app-7c76894bcd         ReplicaSet  cluster  -       ok  -   20s
eduk8s-labs-w02-s045   L.. simple-app-7c76894bcd-rn4sw  Pod         cluster  4/4 t   ok  -   20s
eduk8s-labs-w02-s045  simple-app                       Service     kapp     -       ok  -   20s
eduk8s-labs-w02-s045   L simple-app                    Endpoints   cluster  -       ok  -   20s

Rs: Reconcile state
Ri: Reconcile information

5 resources

Succeeded
```

Figure 11 – *Exemple de commande kapp liée à une application*

l'exemple d'une ligne de commande qui permet de déclencher successivement la configuration, la construction et le déploiement d'une image.

Il convient également de noter qu'en plus de construire des images et de mettre à jour les références, kbld annote les ressources K8s avec les métadonnées d'image qu'il collecte et les rend rapidement accessibles pour le débogage. Très utile lors de l'examen de l'état de l'environnement.

imgpkg : packaging

Imgpkg a deux intérêts principaux : c'est un outil en ligne de commande et également un format de fichier.

Avec la ligne de commande, il est possible de copier, télécharger, déplacer des images dans un autre registre d'images (container image registry comme Harbor par exemple). La **figure 9** en montre un exemple - (9)

imgpkg permet également de générer une image OCI à partir de fichiers arbitraires, appelée **bundle** (voir **Figure 10**). Ainsi imgpkg est capable de copier des bundles et les images qui en découlent dans n'importe quel registre. (10)

L'intérêt de ce bundle est de pouvoir contenir n'importe quel type de fichier : YAML, ytt template, Helm chart, etc. Pouvoir packager des ressources quelle que soit leur origine, permet de les copier dans un environnement privé et/ou sans connexion à Internet (cas très fréquent).

kapp : une alternative à kubectl

Pour déployer une application K8s, la commande habituellement utilisée est kubectl, qui retourne uniquement la liste des ressources affectées au déploiement. Ce que kubectl ne fait pas, c'est lister les modifications apportées afin de donner à l'utilisateur la visibilité sur les changements qui vont s'oppérer ainsi que la possibilité de les confirmer ou de les réfuter. De plus, kubectl ne dispose pas encore d'une capacité de nettoyage pour faire converger un ensemble de ressources lorsque la configuration est réappliquée à un déploiement existant.

L'outil kapp de Carvel compense ces limites de kubectl. Il est conçu en prenant en compte la manière dont Kubernetes conçoit / conceptualise les applications, c'est-à-dire un ensemble de ressources liées, qui partagent une étiquette commune (pod, replicaset, déploiement, services,…).

Figure 11

Voici quelques principes clés du fonctionnement de kapp :

- kapp sépare la phase de calcul du changement (diff) de la phase d'application (apply) pour donner aux utilisateurs une visibilité sur les changements s'appliquant au cluster.
- kapp se fonde sur un libellé unique appliqué aux ressources d'une même application, libérant l'utilisateur du souci de modifier/supprimer d'anciennes ressources lors de la mise à jour de l'application.
- kapp ordonne les ressources afin que le serveur d'API Kubernetes puisse les traiter avec succès lorsque des dépendances existent (par exemple, les CRD et les namespaces sont créés avant les autres ressources).
- kapp essaie d'attendre que les ressources soient prêtes avant de considérer le déploiement comme un succès.

Figure 12

Lors d'un changement de configuration, kapp propose plusieurs fonctionnalités :

- kapp détecte les modifications apportées aux ressources en comparant la configuration locale à la version déjà déployée sur le cluster. Ainsi, kapp ne prend pas en compte l'origine des changements à appliquer et ces changements ne doivent pas obligatoirement être stockés dans le système. Cela signifie que kapp peut être utilisé avec d'autres outils qui produisent des configurations K8s, comme helm ou kustomize.
- kapp peut afficher les modifications dans un diff de style git en fournissant l'indicateur —diff-changes.
- kapp ne réappliquera que les ressources qui ont été modifiées.
- kapp attendra que les pods associés au déploiement soient « prêt » avant de quitter avec succès.

kapp-controller : gestion du cycle de vie

L'API déclarative et l'approche multicouches de kapp-controller permet de construire, de déployer et de gérer vos applications. Cet outil permet également de packager votre logiciel en packages distribués et permet à vos utilisateurs de découvrir, de configurer et d'installer ces packages sur un cluster K8s.
kapp-controller permet de récupérer, de créer des templates et de déployer des applications K8s puis de les garder continuellement à jour quand la configuration des repositories change (gitops/devops), de manière flexible.
Cet opérateur tourne dans un cluster K8s et intègre les outils vus précédemment, ytt, kbld, imgpkg et kapp. De cette manière kapp-controller offre un système robuste et autoportant.
Comme le montre le schéma **Figure 13**, kapp-controller surveille l'évolution du fichier source et, quand un changement est détecté, déploie la dernière version de l'application.
kapp-controller a trois cas d'usages principaux – (11) :

1. La livraison continue en utilisant kapp-controller couplé à ytt et kapp
2. La consommation de package avec les fonctionnalités d'imgpkg
3. La création de package - (12)

Conclusion

Tout comme K8s, Carvel a été développé selon les principes essentiels de la philosophie d'Unix, à savoir, faire UNE chose, mais la faire BIEN. Carvel est donc une boîte à outils spécialisée dans la construction, le déploiement et la mise à jour d'applications sur K8s avec comme vocation d'être combinée avec d'autres outils. Par exemple, certains outils sont utilisés dans Tanzu Application Platform, la plateforme qui simplifie le développement et le déploiement d'applications sur Kubernetes de VMware, et des providers terraform ont également été mis à disposition.
Carvel est un projet open source dont la communauté d'utilisateurs et de contributeurs ne cesse de grandir(13). L'US Army a récemment adopté cette suite(14). Elle est compatible avec les nombreuses régulations que ses équipes IT doivent respecter pour assurer la confidentialité et la sécurité de leurs opérations. Elle leur a permis d'augmenter la productivité des développeurs et des équipes opérations, sans risques, et de gérer facilement de multiples clusters.
N'hésitez pas à suivre l'évolution de Carvel et à participer aux projets.

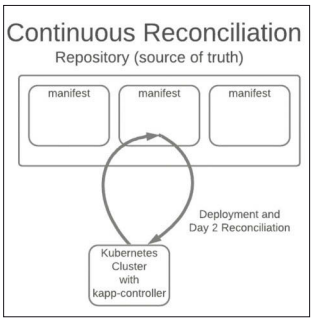


Figure 13 – *Schéma du fonctionnement de kapp-controller kapp-controller a trois cas d'usages principaux*

Figure 12 - *Exemple de sortie après avoir utilisé kapp pour un déploiement*

```
Changes

Namespace                        Name        Kind        Conds.  Age  Op      Wait to    Rs  Ri
eduk8s-labs-w02-s034  simple-app  Deployment  -       -   create  reconcile  -   -
^                                simple-app  Service     -       -    create  reconcile  -   -

Op:      2 create, 0 delete, 0 update, 0 noop
Wait to: 2 reconcile, 0 delete, 0 noop

Continue? [yN]: y

11:17:38AM: ---- applying 2 changes [0/2 done] ----
11:17:38AM: create service/simple-app (v1) namespace: eduk8s-labs-w02-s034
11:17:38AM: create deployment/simple-app (apps/v1) namespace: eduk8s-labs-w02-s034
11:17:38AM: ---- waiting on 2 changes [0/2 done] ----
11:17:38AM: ok: reconcile service/simple-app (v1) namespace: eduk8s-labs-w02-s034
11:17:39AM: ongoing: reconcile deployment/simple-app (apps/v1) namespace: eduk8s-labs-w02-s034
11:17:39AM:  ^ Waiting for 1 unavailable replicas
11:17:39AM:  L ok: waiting on replicaset/simple-app-6b4488bc45 (apps/v1) namespace: eduk8s-labs-w02-s034
11:17:39AM:  L ongoing: waiting on pod/simple-app-6b4488bc45-5mmfr (v1) namespace: eduk8s-labs-w02-s034
11:17:39AM:     ^ Pending: ContainerCreating
11:17:39AM: ---- waiting on 1 changes [1/2 done] ----
11:17:41AM: ongoing: reconcile deployment/simple-app (apps/v1) namespace: eduk8s-labs-w02-s034
11:17:41AM:  ^ Waiting for 1 unavailable replicas
11:17:41AM:  L ok: waiting on replicaset/simple-app-6b4488bc45 (apps/v1) namespace: eduk8s-labs-w02-s034
11:17:41AM:  L ok: waiting on pod/simple-app-6b4488bc45-5mmfr (v1) namespace: eduk8s-labs-w02-s034
11:17:43AM: ok: reconcile deployment/simple-app (apps/v1) namespace: eduk8s-labs-w02-s034
11:17:43AM: ---- applying complete [2/2 done] ----
11:17:43AM: ---- waiting complete [2/2 done] ----

Succeeded
```

(1) https://github.com/vmware-tanzu/carvel
(2) https://carvel.dev/
(3) https://tanzu.vmware.com/developer/workshops/lab-getting-started-with-carvel/
(4) https://github.com/vmware-tanzu/community-edition/blob/main/addons/packages/vsphere-csi/2.4.1/bundle/vendir.yml
(5) https://github.com/vmware-tanzu/community-edition/tree/main/addons/packages/vsphere-csi/2.4.1/bundle/config/upstream/vsphere-csi/manifests/vanilla
(6) https://carvel.dev/ytt/docs/v0.40.0/how-it-works/
(7) https://github.com/bazelbuild/starlark
(8) https://blog.stephane-robert.info/post/Kubernetes-ytt-template-patch/
(9) https://carvel.dev/imgpkg/docs/v0.27.0/air-gapped-workflow/
(10) https://carvel.dev/imgpkg/docs/v0.27.0/
(11) https://tanzu.vmware.com/developer/guides/kapp-controller-gs/
(12) https://carvel.dev/kapp-controller/docs/v0.34.0/packaging-tutorial/
(13) https://github.com/vmware-tanzu/carvel
(14) https://carvel.dev/blog/casestudy-modernizing-the-us-army/

Dorra Bartaguiz

Développeuse .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.

Pourquoi je n'ai plus de classes abstraites dans mon code ?

Depuis quelques années, j'ai constaté que je n'écris plus de classes abstraites dans mon code et je me suis donc posé la question "Pourquoi je n'ai plus de classes abstraites dans mon code ?". En creusant la question, je me rends compte que je n'utilise plus de classe abstraite depuis que j'ai commencé à appliquer le TDD ("Test Driven Development" ou développement dirigé par les tests). J'ai donc décidé d'écrire cet article pour montrer qu'on peut oublier les classes abstraites au profit de la composition.

Je vais d'abord commencer par les définitions de ce qu'est le TDD et qu'est-ce une classe abstraite.

Qu'est-ce que le TDD ?

Kent Beck a créé la méthodologie du développement dirigé par les tests dans les années 2000. « Test Driven Development » est une méthode qui prône une nouvelle gymnastique de l'esprit et une nouvelle discipline : écrire le test avant le code de production. L'idée est donc de se fixer un objectif à atteindre en écrivant un test. Ce dernier permet de savoir quand s'arrêter de développer le comportement attendu. Dans un premier temps, le test sera rouge à l'exécution puisque le comportement du code correspondant n'existe pas encore. On doit donc avoir créé manuellement ou généré via l'IDE la méthode à appeler. La deuxième phase correspond à écrire le minimum de code pour que le test passe au vert. Et enfin la factorisation du code permet de l'améliorer et d'obtenir une meilleure lisibilité et maintenabilité en troisième phase. On parle alors du cycle Red/Green/Refactor comme présenté dans l'image ci-dessous.

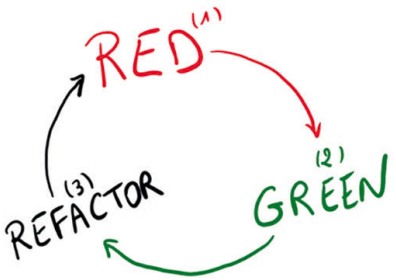
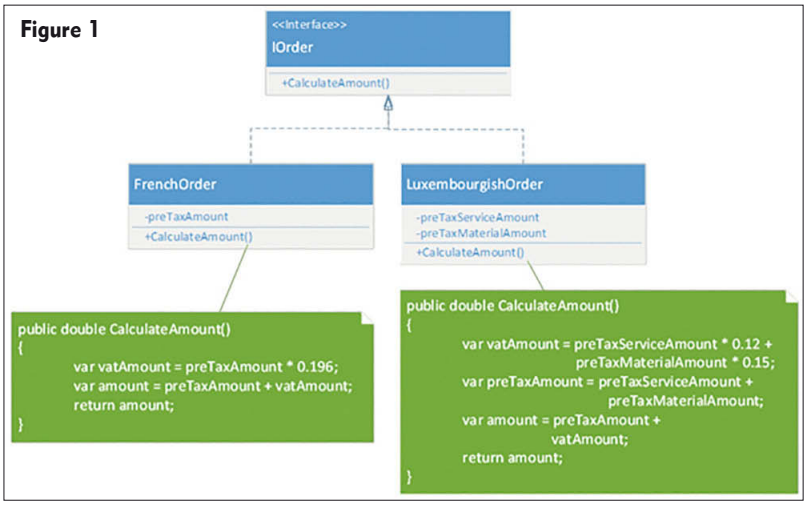


Figure 1



Le développement dirigé par les tests nécessite tout de même quelques prérequis. En effet, avant d'attaquer l'écriture du premier test, il vaut mieux définir les "baby steps". Ce sont de petites étapes pour arriver à la solution finale attendue.

Classe abstraite

C'est une classe dont l'implémentation n'est pas complète (https://fr.wikipedia.org/wiki/Classe_abstraite). Elle peut être héritée, mais ne peut pas être instanciée. On peut y définir une méthode abstraite dont le corps est implémenté dans une classe fille. Elle peut contenir des méthodes ou des attributs implémentés (privés ou publics).

Souvent on l'utilise pour factoriser du code.

Prenons l'exemple suivant : **figure 1**

Le montant total est égal à la somme du montant hors taxe et la TVA. Par contre, la TVA varie selon le pays. Pour la France, elle est de 20% et imaginons que pour le Luxembourg, le calcul est tout autre. Il y a une taxe de 12% sur le service et une taxe de 15% sur le matériel.

Quand on factorise pour créer une classe mère abstraite, on peut définir le calcul du montant total qui est commun aux deux classes filles. Par contre, ce qui est spécifique aux classes filles, on le définit comme abstrait au niveau de la classe mère. Ainsi on a juste à le spécifier au niveau des classes filles. Ci-dessous une solution de factorisation de notre exemple. Elle implémente le « design pattern » (modèle de conception) « Template method ». Ce pattern fait partie de ceux du GoF (Gang of Four) décrit dans le livre « *Design Patterns: Elements of Reusable Object-Oriented Software* » écrit par Erich Gamma, Richard Helm, Ralph Johnson, et John Vlissides, avec une préface de Grady Booch. **Figure 2**

Vue comme ça, la classe abstraite est bien utile pour rendre le code plus lisible et pour bien séparer ce qui est commun de ce qui est spécifique.

Il y a quelques années, c'était la solution que je préconisais surtout après une conception faite comme elle se doit, avant de commencer les développements. En tout cas, c'est ce que j'ai appris à l'école et pendant mes premières expériences en tant que développeuse junior. Et puis, ça paraît propre, lisible et bien factorisé ! D'ailleurs, les promoteurs de l'approche "Big Design Up Front" (BDUF) expliquent que cette approche permet de découvrir les problèmes avant la phase de développement. Donc c'est important pour eux de concevoir toute la solution avant de commencer le développement.

C'est en opposition avec la promotion de l'agilité qui invite à commencer par le MVP (Minimum Viable Product, la version la plus minimaliste et viable du produit) pour expérimenter le produit et ensuite itérer sur les évolutions pour l'améliorer.
En utilisant une classe abstraite, il faut toujours passer par l'instance d'une classe fille pour tester unitairement le comportement commun. On ne peut tester directement le comportement de la classe abstraite vu qu'on ne peut pas l'instancier. Si on veut être sûr·e d'avoir bien respecté le principe de Liskov (L de SOLID : Une classe mère peut être substituée par une classe fille sans que le comportement change), on doit retester le comportement commun à chaque fois qu'on teste une classe fille, puisque le comportement commun ne peut pas être testé directement. On va donc avoir autant de tests que de classes filles pour vérifier que le comportement commun n'est pas modifié. Quand on n'a que deux classes filles, la tâche est simple, mais imaginons le cas d'une dizaine de classes filles, c'est un peu galère, non !

Comment implémenter sans classe abstraite ?

Lors de mes expériences, j'ai rencontré plusieurs façons de développer. Ce qui m'a permis de comparer, d'analyser et de comprendre comment mieux implémenter, même si je suis certaine qu'il y a potentiellement d'autres méthodes et que la mienne est encore perfectible.
Et puis un jour j'ai découvert le principe "Composition over inheritance" dans le livre du GoF cité dans le paragraphe précédent. Ce principe invite à faire de la composition des objets pour implémenter la fonctionnalité voulue. J'ai commencé à l'appliquer dans mes développements. Au début, c'était difficile de s'affranchir de l'héritage comme j'avais l'habitude de l'appliquer et surtout j'étais convaincue de l'utilité de la factorisation avec une classe abstraite. Avec le temps j'ai appris la méthode TDD et je me suis rendue compte que j'utilisais de moins en moins l'héritage.
Reprenons l'exemple du calcul de la taxe. Quand je le fais en TDD et en baby steps, je vais décomposer mon problème en trois parties. Le calcul nominal du montant total, le calcul de la TVA française et celui du Luxembourg.
Pour calculer le montant total, on a besoin de la valeur de la TVA et le montant hors taxe qui vont être récupérés. Et pour chacun des pays, on a le calcul des valeurs de la TVA. Ainsi, on va avoir la classe OrderPrice qui contient la méthode CalculateAmount(). On va créer les tests pour cette méthode. Sauf que pour faire les calculs, on a besoin du montant hors taxe et du montant de la TVA. Pour cela, on crée une interface qui retourne ces deux résultats. À ce stade, on a besoin de la classe à tester (OrderPrice) à laquelle on injecte en dépendance l'interface IVatCalculator (sans implémentation) comme le décrit le schéma suivant. Ainsi, on peut tester unitairement la fonctionnalité de calcul en substituant l'interface injectée. **Figure 3**

En deuxième temps, on va préparer les tests correspondants à l'une des implémentations de l'interface IVatCalculator, suivis par ceux de la deuxième implémentation. En résultat, le diagramme de classes ressemble à ce qui suit. **Figure 4**

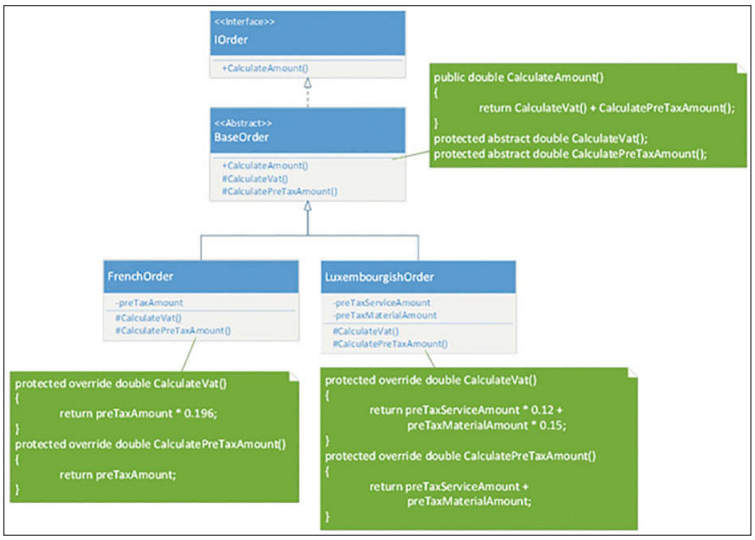


Figure 2

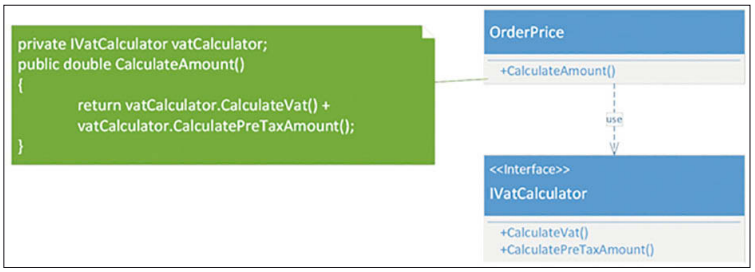


Figure 3

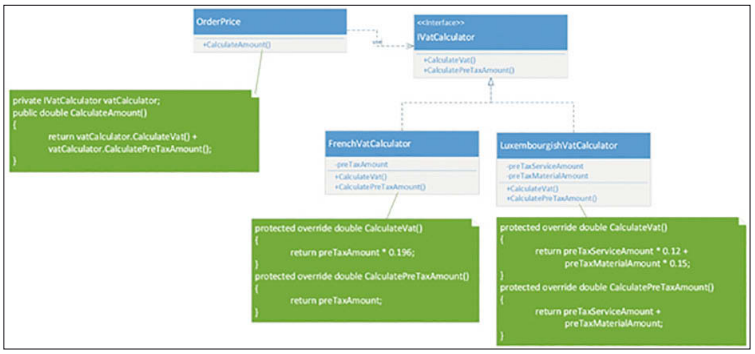


Figure 4

Avant de finir

La démarche que je viens de décrire correspond à l'approche "outside in" en TDD (appelée aussi approche mockiste). Le développement est guidé par le besoin et l'implémentation de la solution se fait de l'extérieur vers l'intérieur en utilisant des substituts appelés aussi mocks.
Robert Martin et Sandro Mancuso ont fait toute une série de vidéos pour expliquer la différence entre les deux approches (outside in Vs inside out) par ici (https://cleancoders.com/series/comparativeDesign).

Conclusion

J'espère que j'ai apporté suffisamment d'informations pour vous montrer qu'on peut se passer des classes abstraites en utilisant la composition. Le code reste tout aussi clair, voire plus clair qu'avec l'héritage. Je considère l'existence d'une classe abstraite dans le code comme un "smell'' (un signe que le code est améliorable) qu'on peut refactorer pour séparer le comportement générique de celui spécifique. Si vous voulez savoir quand refactorer, je vous invite à lire mon article "Quand refactorer et pourquoi" publié dans le n° 251.
N'hésitez pas à revenir vers moi pour en discuter sur les réseaux sociaux (Twitter ou LinkedIn). Je serai ravie de débattre sur le sujet. :)

Michelle Avomo

Michelle est Développeuse FullStack Senior chez CodeWorks. Hétéroclite sur les technos, elle préfère s'éloigner des langages et frameworks et se focaliser sur les pratiques.

Formatrice sur les sujets crafts (entre autres), Michelle est souvent partante pour écrire ou parler des tests.

CodeWorks

Tester ses composants avec la famille dom-testing-library

PUBLIC : PERSONNES AYANT DÉJÀ L'HABITUDE D'ÉCRIRE DES TESTS

Dans les frameworks front-end, un composant est une association entre un template HTML et une classe Typescript ou JavaScript.

En théorie, écrire un test de composant, c'est vérifier de manière automatisée que l'association entre le rendu (états du DOM) et les actions utilisateur produit le résultat attendu. En pratique, implémenter cette vérification dépend fortement des API du framework utilisé.

Dans l'écosystème Vue.js par exemple, *'Vue Test Utils' (VTU)*, la librairie de test officielle définit une API qui permet de reproduire le cycle de vie du composant (*mount, render, destroy, etc.*), de déclencher des actions comme un utilisateur le ferait et de vérifier si l'affichage est conforme. Il n'est pas possible de dissocier la vue (HTML) du modèle (classe JS).

En Angular en revanche, comme il existe trois manières de tester son composant :

- **En isolation *(isolated testing)*** : le test vérifie uniquement le comportement de la classe Typescript. Le DOM est totalement ignoré.
- **En superficiel *(shallow testing)*** : le test associe bien le template et sa classe, mais sans tenir compte de la dépendance vis-à-vis des autres composants.
- **En profondeur *(integrated testing)*** : le test associe le composant et ses dépendances.

Ce n'est que dans le cas des tests en superficiel (shallow testing) et en profondeur (integrated testing) qu'il est nécessaire de s'intéresser à *TestBed*, l'API officielle de tests des composants et de connaître les subtilités de la classe *ComponentFixture*.

Aussi, pour tout développeur front-end écrivant des tests, un changement de framework n'est pas anodin. Il s'agit de devoir apprendre les spécificités de chaque librairie de tests associée aux frameworks front, mais aussi et surtout de renoncer aux API, à leur configuration voire les mécaniques d'optimisation consolidées autour du framework habituel.

Le changement est l'unique constante, particulièrement dans notre métier me direz-vous.

Mais qu'est-ce qu'il y aurait de différent dans l'utilisation d'un composant Angular par rapport à un composant Vue.js ou React d'un point de vue utilisateur ? Le comportement de l'utilisateur devant un composant changerait-il suivant que ce dernier soit écrit en angular, en React ou en Vue.js ? Pourquoi ne pas s'affranchir des détails d'implémentation des frameworks et tester un composant comme le ferait un utilisateur ?

S'affranchir des implémentations propres à chaque framework et écrire des tests de composants conformes aux usages de l'utilisateur : voilà la promesse de la dom-testing-library !

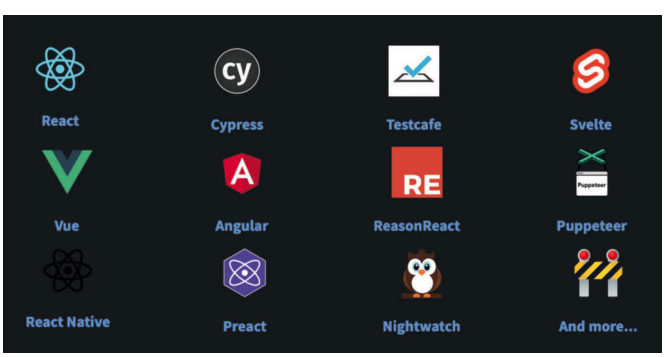


Figure 1

logos des principaux utilisateurs de l'API de la DOM-Testing-Library

Testing-library : la librairie de test qui s'abstrait des nœuds de DOM

Comme pour matérialiser le principe d'inversion des dépendances qui veut que : « *les modules de haut niveau ne devraient pas dépendre des détails. Les deux devraient dépendre des abstractions*», la dom-testing-library en tant que telle définit une « core API ». Cette core API représente le « module de haut niveau».

Chaque framework ou librarie qui le souhaite peut implémenter sa solution en se basant sur la core API et se pose ainsi de facto en « module de bas niveau» ou détail d'implémentation.

À ce jour, selon le site officiel, les librairies et frameworks utilisant la core API de la dom-testing-library sont : React, Cypress, TestCafe, Svelte, Vue, Angular, Puppeteer, React Native, Preact, ReasonReact, NightWatch. **Figure 1**

Il est assez facile de retrouver les déclinaisons associées : *react-testing-library, vue-testing-library, angular-testing-library, testing-library, testCafe-testing-library, etc.*

Quelle conséquence cette déclinaison dans plus dix frameworks cela peut-il avoir pour nous en tant que développeur ? Assurément un gain de temps et une uniformisation des pratiques de tests de composants.

Dans la suite de cet article, nous reviendrons rapidement sur les motivations des auteurs de la librairie puis présenterons les indicateurs de la librairie avant d'illustrer le gain de temps qu'offre la famille *-testing-library par un exemple tests de composants déclinés en Vue.js et en Angular.

Motivation des auteurs de la librairie

En traduisant la présentation depuis le site officiel, dom-testing-library est :

« *(…) une solution très légère pour tester les nœuds du DOM (aussi bien simulé comme avec JSDOM tel que fourni par défaut avec Jest ou présents dans le navigateur). Les principales fonctionnalités qu'elle fournit impliquent l'interrogation du DOM pour les nœuds à la façon dont un utilisateur recherche des éléments dans la page. De cette façon, la librairie permet de s'assurer que vos tests vous donnent confiance dans votre code UI. Le fil conducteur de la librairie DOM Testing est:*

Plus vos tests ressemblent à la façon dont votre logiciel est utilisé, plus ils peuvent vous donner confiance.»

Et pour les personnes qui ne sont pas tout à fait à l'aise sur l'écriture des tests, la famille *-testing-library encourage les bonnes pratiques de tests UI comme nous le verrons dans les exemples plus loin. Une fois les bases posées, vous ne risquez pas de vous en éloigner.

Fiche technique

Au moment de l'écriture de cet article, la dernière version (8.11.0) datait de trois jours et dépassait les six millions de téléchargements hebdomadaires.

Ce qui montre que non seulement c'est un projet maintenu, mais aussi qu'il a rencontré son public et qu'on peut raisonnablement s'y fier.

La librairie s'installe en tant que dépendance de développement.

Fiche NPM

Du fait de multiples déclinaisons vues plus haut, il y a plusieurs résultats liés à testing library sur npm : **figure 2**

Pour obtenir la « core API», la dépendance adaptée est la @testing-library/dom.

En comparaison, Jest, la librairie de test ultra populaire de Facebook a plus de treize millions de téléchargements hebdomadaires.

Fiche state of JS – 2020

Le site stateofjs.com classifie les tendances au sein de l'écosystème JS.

Les auteurs de ce projet interrogent un peu plus de 20 000 développeurs depuis 2016 pour connaître les usages et les opinions des développeurs JS sur leurs outils de travail.

Par défaut, leur classification inclut les quatre indicateurs suivants : **figure 4**

Ainsi, en 2020, la testing-library fait son entrée dans le classement et satisfait 97% des personnes l'ayant déjà utilisé. Cela la place première de la catégorie « outils de tests ». https://2020.stateofjs.com/en-US/technologies/testing/) **Figure 5**

Son taux d'utilisation en revanche n'est que de 26% comme le montre l'image ci-contre. **Figure 6**

Cependant, si au vu de la promesse et des implémentations existantes, vous souhaitez l'introduire dans votre prochain projet, voici un exemple d'implémentation dans deux frameworks.

Fiabiliser les tests d'UI - Cas pratique

Dans cette partie, nous voulons montrer comment la famille *-testing-library fiabilise les tests sur les UI en ne s'intéressant qu'à ce qui est visible à l'écran (text, label, bouton, etc.).

Nous avons voulu montrer comment les opérations de recherche d'éléments du DOM et la gestion des événements diffèrent suivant les librairies de tests officielles et comment les versions *-*testing-library* uniformisent les tests UI.

Pour une meilleure lisibilité, nous nous bornerons aux implémentations en Angular et en Vue.js.

Pour chaque écosystème, nous avons souhaité deux volets :

- La différence d'approche dans les librairies « officielles» et l'approche dom-testing-library

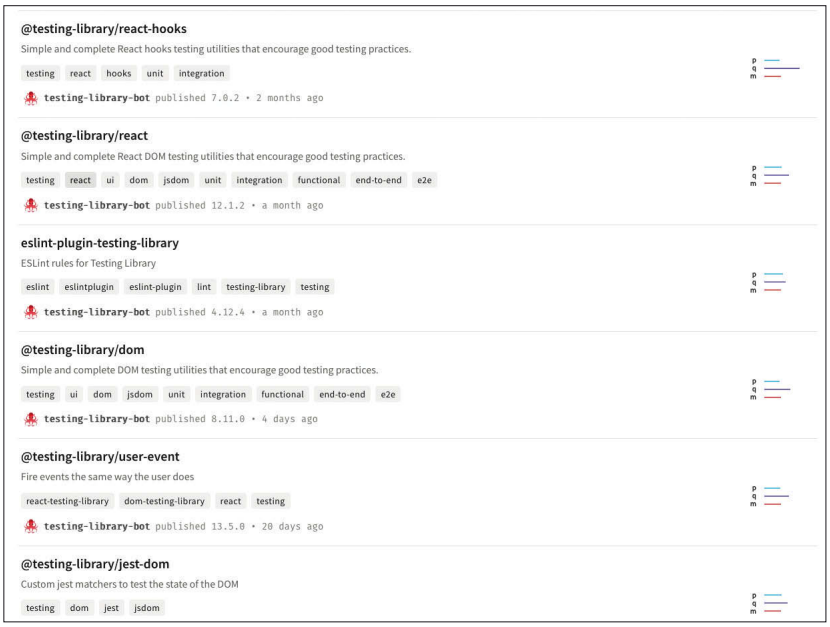


Figure 2 : *extrait de résultats pour une recherche sur le mot-clef "testing-library"*

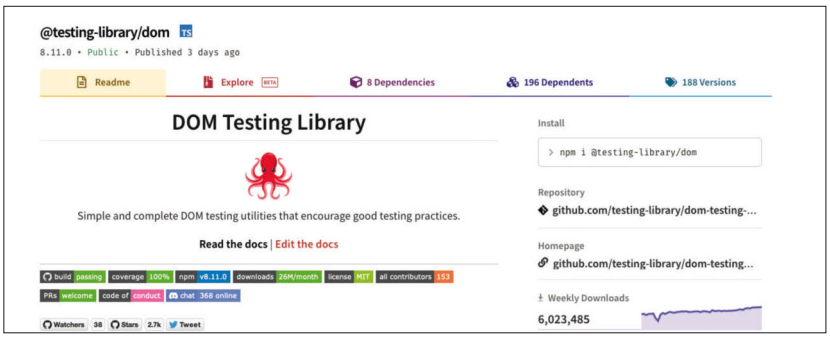


Figure 3: *Fiche NPM de la testing library*

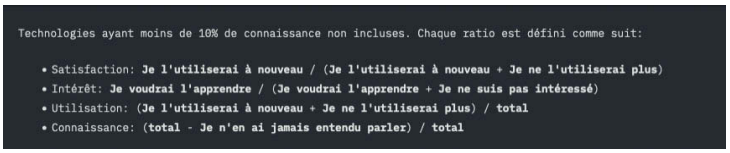


Figure 4

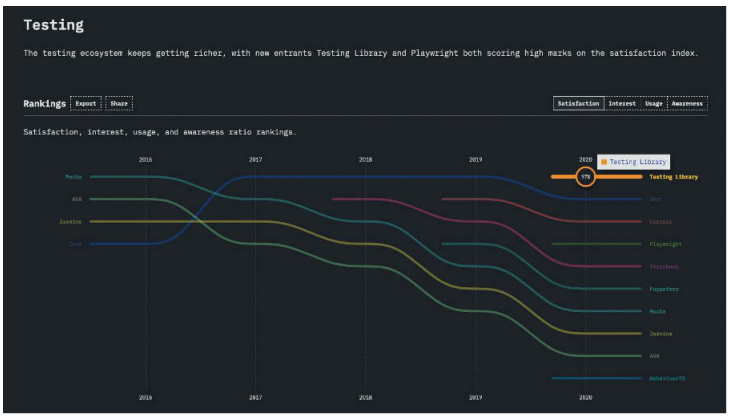


Figure 5 : *Classement « state of JS » des outils de tests par satisfaction*

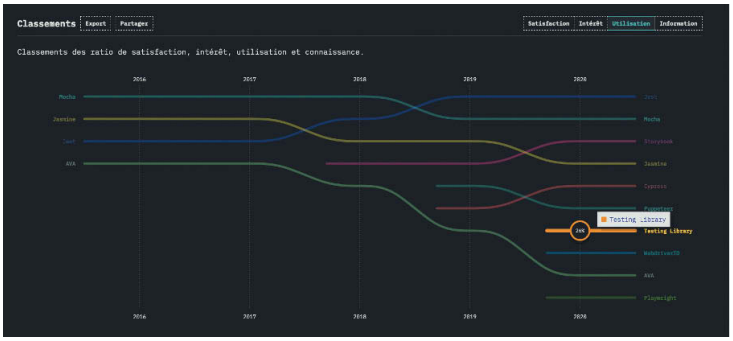


Figure 6 : *Classement « state of JS 2020 » des outils de tests par usage*

```
describe( name: "Official testing lib ", fn: () => {
  it( name: 'render a todo', fn: () => {...})

  it( name: 'create a todo ', fn: async() => {...})
})

describe( name: "Testing library approach ", fn: () => {

  it( name: 'render a todo', fn: () => {...})

  it( name: 'create a todo ', fn: async() => {...})

})
```

Figure 7 :
Organisation des tests

- Montrer comment la dom-testing-library change peu d'un framework à l'autre.

Le besoin métier est d'afficher la liste des choses à faire et d'en rajouter.

Chaque écosystème vérifiera ces deux fonctionnalités à tester comme ci-dessus : **figure 7**

Écosystème Vue.js : Vue Test Utils (VTU) vs Vue-Testing-Library (VTL)

Dans cette partie, nous montrons dans un premier temps comment VTU nous amène à utiliser intensément les "attributs data-* afin de retrouver des éléments du DOM et simuler les actions utilisateur depuis un test de composant.

Dans un second temps, nous montrons comment le même besoin est exprimé avec VTL.

Le code HTML se présente comme ci-dessous :

```
<div>
<div v-for="todo in todos" :key="todo.id" data-test-id="todo" role="list">
 {{ todo.text }}
</div>
<form data-test-id="form" @submit.prevent="createTodo">
 <label for="newTodo">Add new:</label>
 <input data-test-id="add-todo" v-model="newTodo" id="newTodo"/>
</form>
</div>
```

Vue Test Utils (VTU)

Le code associé aux deux cas : affichage d'une liste de tâches et ajout d'une nouvelle tâche.

```
describe("Official testing lib (Vue Test Utils) ", () => {
it('render a todo', () => {
 const wrapper = mount(TodoApp)
 const todo = wrapper.get('[data-test-id="todo"]')

 expect(todo.text()).toBe('Learn testing-library')
})

it('create a todo ', async() => {
 const wrapper = mount(TodoApp)

 expect(wrapper.findAll('[data-test-id="todo"]')).toHaveLength(1)
 await wrapper.get('[data-test-id="add-todo"]').setValue("Another Todo")
 await wrapper.get('[data-test-id="form"]').trigger('submit')

 expect(wrapper.findAll('[data-test-id="todo"]')).toHaveLength(2)
})
})
```

Pour isoler les composants du DOM de la TodoApp à tester, nous avons introduit des attributs *data-** nommés ici ***data-test-id***.

Les attributs data-* permettent de rajouter des infos supplémentaires aux tags HTML5 standard. Ils sont, par définition, techniques donc invisibles pour l'utilisateur.

Nous rappelons ici que les auteurs de la testing-library clament que : *"Plus vos tests ressemblent à la façon dont votre logiciel est utilisé, plus ils peuvent vous donner confiance".*

Dans notre cas, même si les tests sont passants, il reste qu'il y a très peu de chances pour qu'un utilisateur devant son appli TodoApp se dise :

«Pour ajouter une nouvelle tâche, je vais commencer par rechercher la div avec un attribut data-test-id dont la valeur est égale à 'addTodo', puis trouver le bouton dont le data-test-id est égale à 'form' avant de valider le formulaire».

Et pourtant, c'est comme cela que notre test "create todo" est exprimé…

Par ailleurs, au-delà de la charge cognitive liée à la divergence entre l'usage réel d'un utilisateur et la logique très technique des tests censés préserver le bon fonctionnement d'une fonctionnalité, il y a toujours un risque à se baser sur des tags HTML.

L'erreur étant humaine, ce type de tests fonctionne jusqu'à ce qu'une personne supprime ou renomme les attributs data-test-id du template du composant. Cela ne contribue pas à l'augmentation de la confiance en ses tests d'UI.

Qu'en est-il de la version testing-library ?

Vue Testing Library

Le code associé aux deux cas : affichage d'une liste de tâches et ajout d'une nouvelle tâche.

```
describe("Testing library approach ", () => {

it('render a todo', () => {
 const {getByText} = render(TodoApp)

 getByText('Learn testing-library')
})

it('create a todo ', async() => {
 const {getByLabelText, getAllByRole} = render(TodoApp)
 expect(getAllByRole('list')).toHaveLength(1)


 const newTodoInput = getByLabelText(/Add new/i)
 await fireEvent.submit(newTodoInput, 'learn another thing')

 const todos = getAllByRole('list')
```

Figure 8

```
<div class="container">
  <div *ngFor="let todo of todos" id="todo" role="list">
    {{ todo.text }}
  </div>
  <form (submit)="createTodo()">
      <label for="newTodo">Add new:</label>
      <input type="text" id="newTodo" name="newTask" [value]="newTodo" [(ngModel)]="newTodo" [ngModelOptions]="{standalone: true}"/>
  </form>
</div>`,
```

```
    expect(todos).toHaveLength(2)
  })

})
```

Dans cette version, pour reprendre l'implémentation de l'ajout d'une nouvelle tâche (create a Todo), on peut lire :

- trouver le tag dont le texte affiché est "Add new",
- lancer une action de soumission du formulaire en associant une nouvelle tâche au champ dont le label est : "Add new",
- lister tout le contenu associé au rôle 'list'.

Bien que l'on soit loin du langage naturel, il y a déjà dans cette implémentation un effort de ne tester que ce qui est visible par l'utilisateur.

Par ailleurs, la testing-library encourage l'accessibilité et promeut la recherche par rôle comme première option pour trouver un nœud de DOM.

À noter que pour chacune des méthodes de la core API (getByText, getByLabelText, getAllByRole etc.), le test échoue si la valeur en paramètre n'existe pas.

Écosystème Angular : TestBed vs Angular-testing-library

Comme évoqué plus haut, cette partie vise à montrer la différence entre une écriture de tests UI basés sur TestBed et des tests UI écrits grâce à la librairie angular-testing-library.

Le code HTML se présente comme ci-dessus: **figure 8**

TestBed

Avant d'écrire le tout premier cas de test (affichage de la tâche par défaut), nous devons commencer par configurer TestBed et instancier tous les éléments utiles pour la suite des tests. **Figure 9**

Une fois cette initialisation terminée, afin de retrouver les nœuds du DOM du composant à tester qui nous intéresse, nous utilisons la méthode *'querySelector'*.

Pour chaque cas de test concernant le DOM, nous devons impérativement appeler la méthode *"detectChanges"*, sinon l'action de changement n'est pas répercutée dans le DOM. C'est une spécificité Angular.

Pour le reste, nous alternons principalement entre la recherche du nœud et la vérification des actions à déclencher et/ou l'état final du modèle. **Figure 10**

Il est possible d'aller vérifier la nouvelle hiérarchie du DOM, mais, il nous était plus facile de vérifier le modèle. Nous faisons confiance au mécanisme de binding d'Angular qui s'occupera de rajouter la nouvelle tâche dans la liste des tâches.

```
let fixture: ComponentFixture<TodoComponent>
let component: TodoComponent
let debugElement: DebugElement

beforeEach( action: async () => {
  await TestBed.configureTestingModule( moduleDef: {
    declarations: [
      TodoComponent
    ],
    imports: [FormsModule],
    providers: [
      { provide: ComponentFixtureAutoDetect, useValue: true }
    ]
  }).compileComponents();
});

beforeEach( action: () => {
  fixture = TestBed.createComponent(TodoComponent)
  component = fixture.componentInstance
  debugElement = fixture.debugElement
});
```

Figure 9

```
describe( description: 'Official testing lib (TestBed)', specDefinitions: () => {
  let fixture: ComponentFixture<TodoComponent>
  let component: TodoComponent
  let debugElement: DebugElement

  beforeEach( action: async () => {...});

  beforeEach( action: () => {...});

  it( expectation: 'render a todo', assertion: () => {
    fixture.detectChanges()

    const rootNode = fixture.nativeElement
    const todosDivElement = rootNode.querySelector('div[role="list"]')

    expect(todosDivElement.textContent).toBe( expected: ' Learn testing-library ');
  });

  it( expectation: 'create a todo ', assertion: async () => {
   spyOn(component, method: 'createTodo').and.callThrough();
    fixture.detectChanges()

    fixture.whenStable().then(() => {
      const inputTagElement = debugElement.query(By.css( selector: 'input'));
      let inputElement = inputTagElement.nativeElement
      inputElement.value = 'learn another thing';

      const formTagElement = debugElement.query(By.css( selector: 'form')).nativeElement;
      formTagElement.dispatchEvent(new Event( type: 'submit', formTagElement))

      expect(component.createTodo).toHaveBeenCalled()
      expect(component.todos.length).toBe( expected: 2)
    });
  });
});
```

Figure 10

Qu'en est-il du test écrit grâce à la angular-testing-library ?

Angular Testing Library

Figure 11

Toute la mécanique d'initialisation du composant à tester avec TestBed disparaît !
Il reste certes une recherche par *'querySelector'*, mais c'est pour pouvoir déclencher l'action de soumission du formulaire !
Par ailleurs, en mettant côte à côte les tests *-testing-library, les différences sont minimes.

```
describe( description: 'Testing Library approach (Angular)', specDefinitions: () => {

  it( expectation: 'render a todo', assertion: async () => {
    const {getByText} = await render(TodoComponent)

    getByText( container: 'Learn testing-library')
  });

  it( expectation: 'create a todo ', assertion: async() => {
    const {getByLabelText, getAllByRole, container, getByText} = await renderTodoAppComponent()

    let allTodos = getAllByRole( container: 'list');
    expect(allTodos.length).toBe( expected: 1)

    const newTaskInput = getByLabelText( container: /Add new/i)
    const form: any = container.querySelector( selectors: 'form')

    fireEvent.input(newTaskInput, options: {target: {value: 'Learn another thing'}});
    fireEvent.submit(form)

    expect(getByText( container: "Learn another thing")).toBeTruthy()
    const todos = getAllByRole( container: 'list')
    expect(todos.length).toBe( expected: 2)
  });

  function renderTodoAppComponent() {...}
});
```

Figure 11

Cas n°1: Afficher la tâche par défaut

vue-testing-library

```
it( name: 'render a todo', fn: () => {
  const {getByText} = render(TodoApp)

  getByText( text: 'Learn Vue-testing-library')
})
```

angular-testing-library

```
it( expectation: 'render a todo', assertion: async () => {
  const {getByText} = await render(TodoComponent)

  getByText( container: 'Learn angular-testing-library')
});
```

Cas n°2: Ajout d'une nouvelle tâche via le champ du formulaire

vue-testing-library

```
it( name: 'create a todo ', fn: async() => {
  const {getByLabelText, getAllByRole} = render(TodoApp)
  expect(getAllByRole( text: 'list')).toHaveLength( expected: 1)


  const newTodoInput = getByLabelText( text: /Add new/i)
  await fireEvent.submit(newTodoInput, options: 'learn another thing')

  const todos = getAllByRole( text: 'list')

  expect(todos).toHaveLength( expected: 2)
})
```

angular-testing-library

```
it( expectation: 'create a todo ', assertion: async() => {
  const {getByLabelText, getAllByRole, container, getByText} = await renderTodoAppComponent()

  const newTaskInput = getByLabelText( container: /Add new/i)
  const form: any = container.querySelector( selectors: 'form')

  fireEvent.input(newTaskInput, options: {target: {value: 'Learn another thing'}});
  fireEvent.submit(form)

  expect(getByText( container: "Learn another thing")).toBeTruthy()
  const todos = getAllByRole( container: 'list')
  expect(todos.length).toBe( expected: 2)
});
```

Conclusion

La core API de la testing-library apporte un souffle nouveau dans l'outillage du développeur front.
Bien que cet article ait montré uniquement les différences d'implémentation entre Vue.js et Angular, il vous est possible de partir de Cypress ou de React voire de la dom-testing-library en elle même pour n'utiliser que la core API.
L'écosystème front changeant rapidement, miser sur la dom-testing-library comme core skills dès à présent, ne serait pas risqué si vous vous souciez de la fiabilité de vos tests UI.

Bonnes pratiques Node.js

Vous avez décidé de créer une API révolutionnaire avec Node.js. Bravo ! Mais avez-vous choisi Node pour les bonnes raisons ? Quelles sont les conséquences de cette décision ? Comment votre appli va-t-elle évoluer et comment éviter des downtimes ? C'est pour répondre à ces questions que nous sommes ici.

Matthieu Albert

Développeur fullstack js chez CodeWorks. Venant d'un univers Java-Js, il s'est passionné pour le front avec Vue.js puis a élargi ses compétences pour devenir fullstack Js. Il se fera toujours un plaisir de répondre aux questions et aux challenges qui lui sont posés.

CodeWorks

Les avantages et les inconvénients

Node.js vous permet de n'avoir qu'une seule technologie que ce soit pour le front ou le back : JavaScript. Finie la recherche d'un profil idéal qui maîtrise 3 technos : quelqu'un maîtrisant JavaScript peut très vite monter en compétence et être opérationnel en quelques jours.

C'est certes un avantage en termes de recrutement, mais qu'en est-il des avantages techniques ?

Une application Node.js peut être lancée sur n'importe quelle plateforme, tant que vous avez installé Node sur la machine, ce qui laisse la liberté sur le choix du serveur.

Node.js est event-driven. C'est-à-dire qu'il est nativement fait pour réagir à des événements. Un appel à l'API est un événement, l'ouverture d'un fichier est un événement…

Nous avons parlé des avantages, mais qu'en est-il des inconvénients ? Il y en a un principal : Node.js est mono-thread. Nous en reparlerons plus tard, mais il faudra à tout prix éviter les gros calculs pour ne pas bloquer le thread d'exécution de notre app.

Quelques connaissances utiles

Le cycle de release

Node.js a un cycle de release bien défini : il y a une release majeure tous les 6 mois, avec potentiellement des breaking changes. Les releases paires sont maintenues pendant une longue période (30 mois), ce qui permet de les utiliser librement. La dernière release paire est appelée la version "active".

Les versions impaires sont utiles pour préparer le passage à la version paire suivante pour les développeurs de librairies.

Figure 1

Pour éviter d'avoir des problèmes de compatibilité, il faut choisir une version active ou en maintenance paire de Node.js pour tous les environnements (même de dév).

Gestion des erreurs

Les erreurs qui ne sont pas catchées avec un bloc try/catch termineront le processus. Ceci est fait pour éviter que l'application ne se trouve dans un état inconnu et qu'elle soit corrompue. Pour cela, il y a plusieurs choses à mettre en place :

- Utiliser un autre processus qui relancera l'app si elle est down : cela permet de repartir à un état connu de l'application et d'éviter trop de downtimes. Il y a beaucoup d'outils pour mettre ça en place : *upstart*, *forever*, *pm2*… Si votre application est hébergée par un fournisseur, il aura aussi des solutions à vous fournir.
- Monitorer l'application : cela permet de savoir quand l'application a eu des erreurs et surtout de trouver pourquoi. L'objectif étant ensuite de corriger ces erreurs pour éviter au maximum leur reproduction. Pour bien monitorer votre app, utilisez un logger (Winston, Loglevel, npmlog…) et récupérez via un outil (pm2, prometheus + grafana, appmetrics...) les performances de votre app. Cela vous permettra de pouvoir prévenir un potentiel downtime ou un crash.

Il existe un moyen de réagir aux erreurs non catchées : *process.on('uncaughtException')*. Il ne faut pas l'utiliser pour éviter que l'application s'arrête, car elle se retrouverait dans un état corrompu. On peut en revanche l'utiliser pour nettoyer des ressources utilisées par l'application avant qu'elle ne se ferme et donc toujours le terminer avec "*process.exit(errorCode)*" s'il n'y a aucune opération asynchrone en cours ou avec "*process.exitCode = errorCode*" qui permet de ne pas forcer la fin du processus immédiatement.

Mono-thread

Coder avec Node, c'est faire du JavaScript, et JavaScript est mono-thread. Cela signifie que toute opération synchrone bloque le processus qui correspond à votre application. Notez bien que j'ai dit JavaScript : Node.js propose des fonctions synchrone ou asynchrone (ex: *fs.readFile* et *fs.readFileSync*), mais les opérations faites en JavaScript, comme une boucle for, un map, des additions et autres sont elles 100% synchrones.

Si vous voulez savoir si l'opération que vous faites est bloquante ou non, référez-vous à la documentation Node officielle.

Si vous voulez mieux comprendre le côté bloquant ou non du JavaScript, je vous conseille la vidéo de Philip Roberts sur l'event-loop, faite lors d'une JsConf.

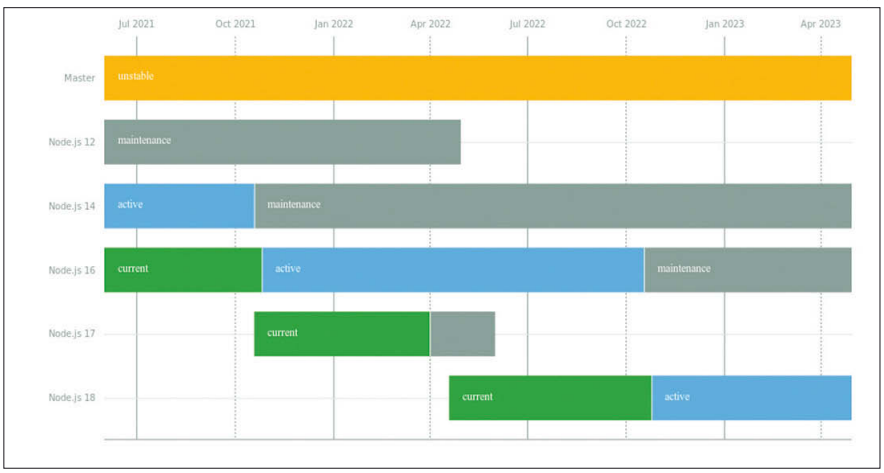


Figure 1 : *Rythme des releases Node.js*

Si vous bloquez le processus de votre application, elle ne réagira plus. Vous avez peut-être déjà vu des sites ou des applis qui ne réagissaient plus au clic. Bloquer votre processus principal aura le même effet.

Documentation Node : https://nodejs.org/api/
Vidéo event loop : https://www.youtube.com/watch?v=8aGhZQkoFbQ

Conséquences du mono-thread

Il faut donc éviter au maximum de bloquer le processus. Mais qu'est-ce qui peut bloquer le processus ? Par exemple :

- des boucles for ou des *map()* sur de très grands arrays, surtout si on les enchaîne aussi avec des *filter()* par exemple. Il ne faut pas oublier que chacune de ces opérations est un parcours complet de l'array.
- des regex complexes. Par exemple, pour une vérification d'email : */^[a-zA-Z0-9][a-zA-Z0-9_\\.\\-\\+&]*@([a-zA-Z0-9]([a-zA-Z0-9]*[\\-]?[a-zA-Z0-9]+)*\\.)+[a-zA-Z]{2,10}$/* qui est de plus en plus longue à résoudre plus l'email est long.

Bref, tout ce qui est une opération complexe est à éviter. Vous pouvez utiliser *performance.now()* avant et après l'opération et faire la différence pour avoir le temps qu'elle prend en millisecondes.

Mais comment faire si on doit passer par une opération complexe même après avoir optimisé son code au maximum ? Il y a quelques solutions :

- Utiliser d'autres programmes écrits dans d'autres langages qui sont multithreadés en les appelant dans notre programme Node. On peut par exemple utiliser le gzip ou le SSL du reverseProxy quand on écrit une API.
- Utiliser les Worker Threads de Node. Ils sont un moyen mis à notre disposition depuis la version 11.7.0 pour exécuter du code sur d'autres threads que le thread principal.

Disons, par exemple, que nous devons calculer un nombre de la suite de Fibonacci. Pour éviter de bloquer le processus principal, nous allons utiliser les Worker Threads :

index.js

```
<pre>
const {Worker} = require("worker_threads")

let num = 40

// Create new worker
const worker = new Worker("./worker.js", {workerData: {num: num}})

//Listen for a new message from worker
worker.once("message", result => {
 console.log(`${num}th Fibonacci Number: ${result}`)
})

worker.on("error", result => {
 console.log(error)
})

worker.on("exit", exitCode => {
 console.log(exitCode)
})

console.log("Executed in the parent thread")
</pre>
```

worker.js

```
<pre>
const {parentPort, workerData} = require("worker_threads")

parentPort.postMessage(getFib(workerData.num))

function getFib(num) {
 if (num === 0) {
  return 0
 }
 else if (num === 1) {
  return 1
 }
 else {
  return getFib(num - 1) + getFib(num - 2)
 }
}
</pre>
```

Ici, nous calculons le 40e numéro de la suite de Fibonacci sans bloquer le processus principal et celui-ci sera affiché grâce au *console.log()* dans *worker.once("message",…)*.

Pour le cas des API, vous pouvez aussi utiliser un Cluster qui crée des threads partageant les ports du serveur. Le processeur principal servira alors juste à rediriger les connexions utilisateurs vers le bon thread.

Cluster : https://nodejs.org/api/cluster.html#cluster

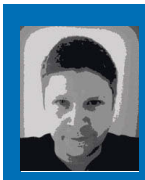
Worker threads :
https://nodejs.org/api/worker_threads.html#worker-threads

Code : https://github.com/amatthieu/worker-thread-demo

Liste de bonnes pratiques :
https://github.com/goldbergyoni/nodebestpractices

Découverte de myQLM et son langage AQASM, la solution quantique d'Atos

Les différentes solutions de simulation quantique, Microsoft Quantum (et son langage Q#) et IBM Qiskit notamment, ont certainement éclipsé quelques autres démarches. En particulier la plate-forme myQLM, développée par l'entreprise Atos et à ce titre solution française, ne fait que très peu l'objet d'articles dans la presse spécialisée informatique ou scientifique. C'est tout l'objet du présent texte que de défricher cette solution et de permettre autant que possible de l'expérimenter.



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 2019 des cours d'informatique quantique auprès d'élèves ingénieurs de l'ECE Paris.

Description de la solution

L'idée pour Atos est de démocratiser l'apprentissage des technologies quantiques, en proposant la possibilité de créer des circuits quantiques avec le langage **AQASM** (*Atos Quantum Assembler*) dont l'utilisation est facilitée par le recours à une librairie Python nommée **PyAQASM** (Python Atos QASM). La programmation se réalise donc essentiellement en Python. L'exécution en local se réalise par un simulateur qui selon la machine courante peut atteindre 20 qubits. Techniquement le code AQASM envoyé vers le simulateur local peut prendre deux formats :

- le format **.aqasm**, qui est un format texte au contenu intelligible ;
- le format **.circ** auquel la ligne de commande a recours quand cette possibilité est utilisée pour l'activation de la simulation.

La solution quantique d'Atos permettra à terme de solliciter des ordinateurs quantiques, bien réels cette fois, qui prendront justement l'un de ces deux formats en entrée. Le langage AQASM est donc le format d'entrée de référence des machines quantiques actuelles et futures de l'entreprise Atos. L'objectif pédagogique est donc doublé d'un objectif industriel de proposition d'une architecture et d'un langage de référence associé, en ce qui concerne le quantique chez Atos.

Enfin, mentionnons un atout singulier de la plate-forme : l'**interopérabilité** avec d'autres plateformes de simulation quantique : Qiskit, ProjectQ, Cirq, et PyQuil. Ainsi vous pouvez écrire votre programme quantique en AQASM puis grâce à cette passerelle, l'exécuter sur un autre simulateur quantique, un simulateur de Qiskit en ligne par exemple.

La documentation en anglais est disponible en ligne ; elle inclut tous les aspects abordés ici : installation, programmation quantique, interopérabilité (notamment avec Qiskit), ressources dédiées à la recherche opérationnelle :
https://myqlm.github.io

Installation de myQLM

La programmation autour de myQLM utilise largement les notebooks **Jupyter**. C'est-à-dire qu'usuellement on conçoit nos programmes quantiques sous Jupyter. On peut également lancer la simulation de ces programmes directement depuis Jupyter. Une fois mis au point sous Jupyter, on peut évidemment exporter notre programme en un fichier .aqasm ou .circ pour utilisation éventuelle sur une machine quantique réelle.

Il est assez classique d'installer la distribution **Anaconda** pour obtenir Jupyter (en plus d'autres logiciels Python à vocation scientifique). Il est également possible d'utiliser le gestionnaire de paquets Python **pip** en ligne de commande.

```
pip install jupyter
```

On installe à présent myQLM, également avec pip.

```
pip install myqlm
```

Pour générer les images représentant les circuits quantiques, en particulier dans un notebook Jupyter, il vous faudra myQLM Magics. La ligne de commande suivante permet de l'installer.

```
python -m qat.magics.install
```

Premier programme AQASM avec Jupyter

L'ensemble de ce qui suit est réalisé dans un notebook Jupyter, disponible en ligne à l'adresse suivante :
https://github.com/benprieur/TESTAQASM/blob/main/TestAQASM.ipynb
Disponible dans ce dépôt Git :
https://github.com/benprieur/TESTAQASM

On travaille d'emblée avec un notebook Jupyter. L'idée est ici de produire un des quatre états de Bell, c'est-à-dire un des états d'intrication maximale. En l'occurrence, nous allons essayer de produire cet état :

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

On commence par créer le circuit proprement dit en AQASM. On commence par importer les objets dont on aura besoin.

```
from qat.lang.AQASM import Program, H, CNOT
```

On procède à la création du programme quantique avec deux qubits en entrée.

```
qprog = Program()
nbqbits = 2
qbits = qprog.qalloc(nbqbits)
```

On ajoute une porte de Hadamard (H) sur le premier qubit,

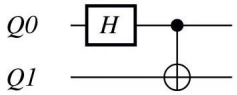
puis une porte CNOT entre la sortie de la porte de Hadamard et le second qubit. Notre circuit est prêt.

```
qprog.apply(H, qbits[0])
qprog.apply(CNOT, qbits[0], qbits[1])
circuit = qprog.to_circ()
```

On procède à la génération graphique de la représentation du circuit quantique.

```
%qatdisplay circuit
```

On obtient alors la représentation graphique suivante.



Le circuit quantique étant prêt, on procède à son utilisation. On importe la classe PyLinalg que nous allons tout de suite utiliser.

```
from qat.qpus import PyLinalg

pylinalgqpu = PyLinalg()
job = circuit.to_job()
```

On lance à présent la simulation quantique.

```
result = pylinalgqpu.submit(job)
```

On analyse les résultats sur les deux qubits de sortie et on mesure l'amplitude de chacune des sorties.

```
for sample in result:
  print("état %s amplitude %s" % (sample.state, sample.amplitude))
```

On obtient ceci comme résultat.

Résultat

état |00> amplitude (0.7071067811865475+0j)
état |11> amplitude (0.7071067811865475+0j)

Ce résultat correspond bien à :

$$\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}$$

On peut à présent procéder à une simulation plus ambitieuse en sollicitant le circuit quantique un millier de fois, pour vérifier que les probabilités obtenues sont bien celles correspondant aux états de Bell.

On crée un job de 1000 essais.

```
job = circuit.to_job(nbshots=1000)
```

Puis on l'applique à notre circuit quantique.

```
result = pylinalgqpu.submit(job)
```

Puis on analyse les résultats après mesure quantique.

```
for sample in result:
  print(sample.state)
  print(sample.probability)
```

On obtient le résultat suivant en sortie (le résultat est probabiliste et peut donc varier d'un essai à l'autre) :

Résultat

|00>
0.478
|11>
0.522

Cela signifie que |00> et |11> sont les deux seuls types de résultats obtenus et qu'ils sont presque équiprobables (478 essais qui donnent |00> et 522 donnent |11>). On obtient aucun essai donnant |01> ou |10> ce qui correspond effectivement à l'intrication maximale recherchée.

Interopérabilité avec d'autres solutions quantiques

Chose rare dans le secteur des plates-formes, AQASM propose de l'interopérabilité vers d'autres solutions. Les cinq solutions supportées sont : Qiskit, OpenQASM, PyQuil, ProjectQ et Cirq. Pour au moins trois de ces solutions, l'interopérabilité fonctionne dans les deux sens. Ainsi pour Qiskit, on peut :

- Concevoir un circuit quantique avec AQASM et le simuler sur Qiskit.
- Réciproquement, on peut exécuter sur AQASM, un circuit quantique conçu avec Qiskit.

Un tel usage nécessite d'installer un module supplémentaire. Pour Qiskit, on exécute cette ligne de commande dans le terminal.

```
pip install myqlm-interop[qiskit_binder]
```

Ensuite l'essentiel de cette interaction bidirectionnelle consiste en la conversion d'un circuit AQASM en un circuit Qiskit ou réciproquement de Qiskit vers AQASM. Les deux petits codes suivants assurent respectivement chacune des deux conversions.

Qiskit vers AQASM :

```
from qat.interop.qiskit import qiskit_to_qlm
qlm_circuit = qiskit_to_qlm(your_qiskit_circuit)
```

AQASM vers Qiskit :

```
from qat.interop.qiskit import qlm_to_qiskit
qiskit_circuit = qlm_to_qiskit(your_qlm_circuit)
```

Fort de notre circuit converti en Qiskit, on peut le simuler sur une plate-forme Qiskit. Il nous faudra seulement déclarer notre *token* IBM (récupérable sur le site IBM en ligne : "API token") ainsi que le nom du simulateur visé (nous utiliserons ici le simulateur nommé *ibmq_qasm_simulator*). On commence par déclarer les modules et classes dont on aura besoin.

```
from qat.interop.qiskit import qlm_to_qiskit
from qat.interop.qiskit import BackendToQPU
```

Puis on crée un job avec 333 essais sur notre circuit en AQASM.

```
job = circuit.to_job(nbshots=333)
```

On convertit alors notre circuit myQLM / AGASM en un circuit IBM Qiskit.

```
qiskit_circuit = qlm_to_qiskit(circuit)
```

On déclare alors notre token de l'API IBM ainsi que le simulateur IBM que nous allons utiliser (en précisant la manière de solliciter ce simulateur de façon synchrone, comme ici, ou asynchrone).

```
MY_IBM_TOKEN = "xxxxxxxxxxxxxxxxxx"
qpu = BackendToQPU(token=MY_IBM_TOKEN,
ibmq_backend="ibmq_qasm_simulator")
```

Puis on soumet le circuit converti à une machine quantique IBM ; dès que possible, on analyse le résultat transmis :

```
result = qpu.submit(job)
for sample in result:
  print(sample.state)
  print(sample.probability)
```

On obtient le résultat suivant. Il est cohérent avec le résultat souhaité (intrication maximale) ainsi qu'avec les résultats obtenus précédemment sur myQLM :

Résultat

|00>
0.5045045045045045
|11>
0.4954954954954955

Pour aller plus loin : théorie des graphes

La solution myQLM compte un certain nombre de ressources qui permettent d'utiliser des algorithmes quantiques appliqués à des problèmes d'optimisation en recherche opérationnelle, en particulier relatifs à la théorie des graphes. La présente section détaille donc différents problèmes proposés dans la documentation en ligne ainsi que les ressources associées dans myQLM à ce propos.

Documentation en ligne à propos de théorie des graphes :
https://myqlm.github.io/advanced_combinatorial_optimization.html#

Problème de coupe maximum

Si on considère un graphe donné, chaque arête a une valeur donnée. Le problème de coupe maximum consiste en trouver la coupe qui traverse la somme maximale de valeurs d'arêtes. Ce problème a un certain nombre d'applications concrètes, notamment le modèle d'Ising en physique, consistant à rechercher les états d'énergie minimale (dans ce modèle les sommets du graphe sont des atomes). Ce problème revient à maximiser le partitionnement en deux sous-graphes du graphe principal. Dans le notebook Jupyter ci-dessous, l'exemple pris ne comprend aucune valeur sur les arêtes. Il s'agit donc de trouver une coupe maximisant le nombre d'arêtes dans la coupe :

Notebook Jupyter

https://mybinder.org/v2/gh/myQLM/myqlm-notebooks/HEAD?filepath=tutorials%2Fcombinatorial_optimization%2Fmax_cut_myqlm.ipynb

Quelques éléments utiles du code AQASM pour ce problème donné. On utilise le module Python networkx pour définir puis matplotlib pour dessiner le graphe :

```
import networkx as nx
import matplotlib.pyplot as plt
```

Ensuite on utilise la class MaxCut pour énoncer le problème de coupe maximum :

```
from qat.opt import MaxCut
max_cut_problem = MaxCut(graph)
```

Enfin, on peut créer un job et lancer une résolution quantique sur myQLM :

```
from qat.qpus import SimulatedAnnealing
from qat.simulated_annealing import integer_to_spins
from qat.core import Variable
...

sa_qpu = SimulatedAnnealing(temp_t=temp_t, n_steps=n_steps)
problem_job = max_cut_problem.to_job(tmax=tmax)
problem_result = sa_qpu.submit(problem_job)
```

Partitionnement de graphes

Le problème précédent (coupe maximum) n'est qu'un exemple parmi d'autres du problème général de partitionnement de graphes selon un critère donné. L'idée est ici d'extraire de façon optimisée des sous-graphes d'un graphe principal. Là encore, on a une classe dédiée pour énoncer ce problème, une fois le graphe principal défini.

```
from qat.opt import GraphPartitioning
graph_partitioning_problem = GraphPartitioning(graph, A, B=B)
```

Un exemple est disponible dans le notebook Jupyter suivant :

Notebook Jupyter

https://hub.gke2.mybinder.org/user/myqlm-myqlm-notebooks-u0tgo7py/notebooks/tutorials/combinatorial_optimization/max_cut_myqlm.ipynb

Problème de la clique

Ici on recherche les sous-graphes complets d'un graphe principal. C'est ce sous-graphe que l'on appelle justement une clique : un sous-graphe dont deux sommets quelconques sont toujours adjacents. La classe KClique permet ici d'énoncer le problème avant de solliciter un simulateur quantique.

```
from qat.opt import KClique
kclique_problem = KClique(graph, K, A, B)
```

Un exemple est disponible dans le notebook Jupyter suivant :

Notebook Jupyter

https://hub.gke2.mybinder.org/user/myqlm-myqlm-notebooks-2qee4rap/notebooks/tutorials/combinatorial_optimization/k_clique_myqlm.ipynb

Problème de couverture par sommets

Dans ce problème on cherche à trouver l'ensemble minimal de sommets qui permet de couvrir toutes les arêtes (et donc tout le graphe). Là encore, une classe est dédiée à la modélisation du problème avant simulation quantique. Il s'agit de la classe VertexCover.

```
from qat.opt import VertexCover
vertex_cover_problem = VertexCover(graph, A=A, B=B)
```

Un exemple est disponible dans le notebook Jupyter suivant :

Notebook Jupyter

https://hub.gke2.mybinder.org/user/myqlm-myqlm-notebooks-ac9aech3/notebooks/tutorials/combinatorial_optimization/vertex_cover_myqlm.ipynb

Conclusion

La solution myQLM d'Atos, moins popularisée que Qiskit d'IBM par exemple, présente pourtant de nombreux atouts sans doute trop méconnus (en plus d'être une solution technologique d'origine française et européenne) :

- Une programmation en Python.
- Une interopérabilité forte avec d'autres plateformes comme abordé ici avec Qiskit.
- Un socle solide de classes dédiées à des problèmes réputés NP-difficiles ou NP-complets, utilisés massivement en sciences et dans l'industrie.
- Une documentation en ligne très concrète incluant de nombreux notebooks Jupyter disponibles.

Directives de compilation

PROGRAMMEZ!

Programmez! n°252 - Mai - Juin 2022

Seigneur Sith niveau directeur des rédactions :
François Tonic - ftonic@programmez.com

Review : **Dorra Bartaguiz**

Contacter la rédaction : redaction@programmez.com

Ont collaboré à ce numéro : **Louis Adam (Zdnet)**
CommitStrip

Les contributeurs techniques

Jérémy Bruyère & Léo Lhuile
Fanny Klauk
Julien Leborgne
Pierre Germain-Lacour
Thierry Fondrat
Gagoba Koffi Narcisse
Gaëtan Viola-Nguyen
Christophe Heral
Cynthia Henaff
Sylvain Cuenca
Jean-Paul Gallant
Maxime Beugnet
Alexandre Caussignac
Elise Savornin
Dorra Bartaguiz
Michelle Avomo
Matthieu Albert
Benoît Prieur

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

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 **Sylvie**, en charge du recrutement chez SoftFluent :

06 67 14 01 39

sylvie.benjelloun@softfluent.com

- Conseil
- Expertise
- Partage

softfluent.fr