

PROGRAnNEZ!

Le magazine des développeurs

N°250
01/02
2022

LES FEMMES ET LA PROGRAMMATION : LANCE-TOI !

NESTJS
JAVASCRIPT
PYTHON
TEST
API
POSTGRESQL
ACCESSIBILITÉ
SÉCURITÉ
GO
BOT & NO-CODE



photo : © DISNEY - TRON

| UN | NUMÉRO | CODÉ | COMPIÉ
| PAR | POUR | LES DÉVELOPPEUSES
| ET | LES | DÉVELOPPEURS

M 04319 - 250 - F: 6,99 € - RD



#250

“Tu veux t'élever au rang de Chevalier ou de Maître ? Intéresse-toi au craft, jeune Padawan !”



En choisissant, un parcours informatique, que ce soit en parcours universitaire classique ou en reconversion, on apprend souvent le hardware et le software. Le hardware tourne autour des machines, leur composition, le câblage... En software on va se tourner vers les langages de programmation, la communication réseau, les

interfaces graphiques. Au final on apprend comment développer un site web, une application mobile ou desktop. En face du nombre infini de langages (ou quasiment infini) souvent on se spécialise, on choisit. On se laisse séduire par un ou deux langages et on essaye d'avoir de l'expérience autour de leurs écosystèmes. Avec l'expérience, on devient spécialiste front (pour ceux qui développent du web) ou spécialiste back (pour ceux qui développent des services) ou alors on est touche-à-tout en étant fullstack (le profil recherché partout). Notre métier de développeur évolue, de nouveaux besoins émergent, on cherche à être efficace, rapide avec toujours une qualité au top. De nouveaux rôles voient le jour avec les profils devops pour rapprocher l'équipe infra à l'équipe dev, les profils data pour explorer les masses de données obtenues des utilisateurs. Ces profils même s'ils ne sont pas appelés développeurs, développent quand même des algorithmes que ce soit pour automatiser les livraisons et les setups des machines côté devops ou pour exploiter les données côté équipe data. La différence réside dans le langage utilisé et la finalité d'utilisation du programme réalisé. En fin de compte, à partir du moment où on veut automatiser un traitement, on est développeur ou développeuse. Quel que soit le nom du profil, on est tous attendu en terme de qualité du livrable.

Il y a 10 ans en France, un mouvement a vu le jour avec la communauté Craft (Software Crafters). Le craft, ou l'artisanat, est un ensemble d'attitudes et d'aptitudes qui permettent de conjuguer entre apporter de la valeur le plus rapidement possible aux utilisateurs et garder un niveau de qualité élevé. Ça ne s'arrête pas à des méthodes telles que le TDD ou BDD ou encore DDD. Oui, les dévs adorent les acronymes et abréviations. Même si ces méthodes ont fait leurs preuves et

continuent à être utilisées, une minorité de développeurs l'applique au quotidien et les "crafters" peinent à évangéliser à ces bonnes pratiques. Effectivement, leur usage nécessite un changement de mindset complet des développeurs, ne plus penser technique mais plutôt métier. Garder les choix techniques après le développement du cœur du métier est un changement radical pour un développeur ou une développeuse à qui on a appris que le choix technique est indispensable dans un projet. Comment veut-on viser la qualité si on n'applique pas le craft quotidiennement ? Où est la place du craft dans le développement d'aujourd'hui et comment faire en sorte que ça se réponde encore plus ? Je pense qu'il y a plusieurs réponses et plusieurs axes d'amélioration. D'un côté, je pense qu'il faut revoir le programme universitaire pour y mettre plus d'ateliers pratiques avec des crafters, apprendre aux étudiants à collaborer, non pas en se partageant les tâches et chacun fait dans son coin, mais bien en co-crédant et en co-écrivant le code via le pair-programming par ex. Côté conférences aussi, il y a du boulot pour rendre plus pragmatique le contenu des talks qui sont généralement axés sur la promotion des outils et langages. Bref, pour devenir un meilleur développeur ou une meilleure développeuse, il faut penser à s'améliorer au niveau du langage et des outils mais surtout comprendre le métier et les besoins fonctionnels. Donc jeune padawan qui tu es, penche toi sur ton métier et le besoin fonctionnel avant de parler technique. La compréhension et la maîtrise du métier sont les clés pour viser l'excellence dans le métier de dev. En ce début d'année, on vous propose un numéro un peu particulier, écrit exclusivement par des femmes du milieu informatique. Même si Programmez! publie régulièrement des articles de femmes dans ses numéros, on a voulu mettre à l'honneur ce rôle rare dans notre milieu par un numéro exclusivement féminin y compris l'édito ;) Donc merci à toutes les auteures d'y avoir participé. Le succès de cette initiative est tel, qu'on a dû sélectionner des articles pour ce numéro et les autres seront publiés dans de prochains numéros. Mon vœux pour cette nouvelle année serait d'avoir encore plus de femmes qui osent dans notre métier, des rôles modèles pour d'autres, des leaders féminines.

Bonne année 2022 !
Dorra Bartaguiz

Contents

Agenda	4
Les brèves	6
De maman au foyer à développeuse web	8
Reconversion réussie et carrière	10
Quand développer (re)devient une évidence	13
Si tu n'es pas passionnée, tu n'es pas une bonne développeuse ?	15
Le métier de développeur.se est sursollicité	17
Coaching Agile, regards croisés	20
Bootcamp de 3 mois et l'expérience que j'en retire	22
Les enjeux de la féminisation des métiers du numérique	24
Designeuse, j'ai fait de l'humanitaire chez mon client	26
Approval testing : l'art de sécuriser la refonte de son legacy	30
Comprendre RXJS	34
Dehors, le monde est rempli de dragons !	37
RTK Query	40
Abonnement & boutique	42 & 43
Ruby : 3 indispensables	47
NestJS	48
Il serait peut-être temps de changer de mot de passe...	53
Touche pas à mon code	56
Pourquoi j'ai voulu faire de l'accessibilité	59
Chercher l'aiguille dans les données	62
Tester une API REST sans tomber dans les pommes (d'API)	66
Poème	68
Maîtriser les interfaces en Go	69
Un bot nocode sur la Google Home	71
Comment configurer PostgreSQL pour la production quand on est débutant.e ?	73
Abordez le rebase...	75
Gestion des logs en Python	78
Le strip du mois	82

PROCHAINS NUMÉROS

Programmez! Spécial
100 % jeux vidéo
11 février 2022

Programmez! n°251
4 mars 2022
JavaScript,
Quantique, Ruby

Les événements Programmez!

Meetups Programmez!

Les dates du 1er semestre :

2 février, 8 mars, 5 avril, 10 mai, 28 juin

Où : Devoteam 43 bd Barbès, Paris

Métros : Château Rouge (ligne 4)

A partir de 18h30

DevCon Serverless + Infrastructure as Code + Kubernetes

17 mars

Où : Campus de l'école 42

96 bd Bessières, Paris

Transport : Porte de Clichy (REC C, ligne 13, ligne 14)

T3b : arrêt Honoré de Balzac

A partir de 13h30

INFORMATIONS & INSCRIPTION : [PROGRAMMEZ.COM](https://programmez.com)

Janvier 2022

					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
				Touraine Tech (Tours)		
24	25	26	27	28	29	30
31						

Février 2022

	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28						

A VENIR

- DevOxx France : du 20 au 22 avril / Paris
- Android Maker : 25-26 avril / Paris
- AFUP Day 2022 : 20 mai / Lille
- MixIT : 24-25 mai / Lyon
- DevFest Lille : 10 juin / Lille
- JFTL 2022 : 13 & 14 juin
- SunnyTech : 30 juin & 1er juillet / Montpellier
- DevFest Nantes : 20-21 octobre / Nantes

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



27/01
2022

Une conférence en ligne mixant
Produit et Craftsmanship

Débats ouverts et talks autour de

« comment faire pour concevoir le bon produit et bien le réaliser ? »

Avec la présence de :



Avec le code promo : **NEXTPROGRAMMEZ**, obtenez - 60% sur votre place
Plus d'informations et inscriptions : next.zenika.com

Joyeux Noël et Log4Shell pour tous !



Pour Noël, les administrateurs système auront le droit de se faire de nouveaux cheveux blancs : une faille a été découverte dans le projet open source Log4J, permettant aux attaquants d'exécuter du code sur la machine. Un correctif a rapidement été proposé, mais Log4J est victime de son succès : le projet open source est incorporé dans de nombreuses solutions et applications, qui l'utilisent pour mettre en œuvre de la journalisation. La balle est donc dans le camp des éditeurs logiciels qui utilisent le projet open source. Encore faut-il qu'ils en aient conscience.

Nadi Bou Hana : Goodbye Dinum

Le directeur de la direction interministérielle du numérique (Dinum) a annoncé qu'il quitterait son poste en janvier 2022. Nadi Bou Hana, qui avait pris la relève d'Henri Verdier à ce poste en 2018, ne part pas vraiment avec les honneurs : deux enquêtes publiées ces derniers mois par Acteurs Publics et Le Monde mettent en cause son management et les réorganisations orchestrées par le nouveau directeur. Il blâme de son côté « des tentatives de déstabilisation depuis deux ans ». Pour l'instant, son successeur n'a pas été annoncé.

Le cyberscore approuvé par l'Assemblée

Les parlementaires veulent imposer aux opérateurs de plateforme en ligne et aux fournisseurs de solution de communication en ligne l'affichage d'un « cyberscore ». Ce dispositif vise à informer les utilisateurs du service du niveau de sécurisation du service et des données transférées par ce biais. Les détails de cette

Rachat d'ARM : la FTC s'en mêle

Le rachat d'ARM par Nvidia a depuis longtemps du plomb dans l'aile, mais les États-Unis viennent de mettre un sérieux coup d'arrêt à la procédure de rachat initiée en septembre 2020 par le géant des cartes graphiques. Celle-ci avait déjà annoncé au mois de novembre qu'elle examinait de près le rachat proposé par ARM, avant de passer

mesure restent encore à déterminer, il s'agit notamment de savoir exactement quels critères doivent être pris en compte par ce cyberscore et qui sera chargé de valider sa fiabilité, mais l'Assemblée comme le Sénat ont validé le principe de la mesure, qui devrait donc être promulguée une fois que les deux chambres auront trouvé un terrain d'entente.

Pénurie de semi-conducteurs : vers un European Chips Act

La pénurie de semi-conducteurs a donné des sueurs froides à la Commission européenne, qui entend maintenant proposer un plan au début de l'année 2022 afin de soutenir la filière et l'innovation sur la production de semi-conducteurs en Europe. Baptisé European Chips Act, le texte espère doubler la production de semi-conducteurs en Europe, afin d'atteindre 20 % de la production mondiale et prévoit un mécanisme de « préférence européenne » en cas de crise. Et c'est tout ce que ça vous fait quand on vous dit qu'on va produire des Chips ?

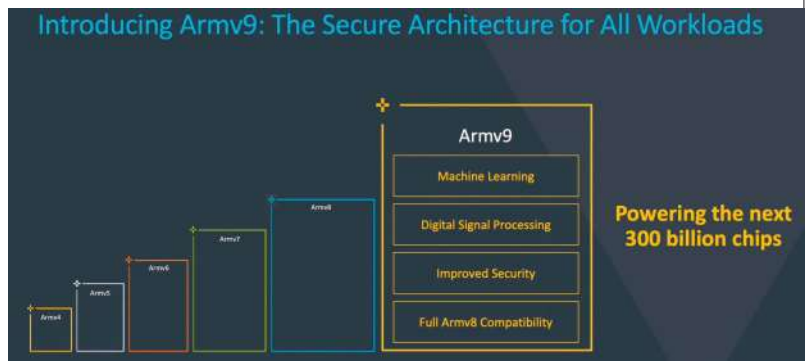
Ransomware : une épidémie qui dure, et LDLC en fait les frais

Le vendeur de matériel

informatique LDLC a eu le malheur de passer dans le viseur du groupe de ransomware Ragnar Locker. La société a publié un communiqué faisant état d'une attaque informatique ayant touché son système informatique, tout en assurant que les données des utilisateurs n'avaient pas été touchées. Le groupe a de son côté revendiqué l'attaque sur son site, publiant au passage environ 30 Go de données volées et des screenshots montrant les équipes de LDLC en pleine cellule de crise pour répondre à l'incident. Une provocation dont le vendeur se serait bien passé, mais les groupes de ransomware ne font pas vraiment dans la finesse.

Jack Dorsey fait ses adieux à Twitter

Il est courant de voir des utilisateurs de Twitter annoncer à grands cris qu'ils quittent le réseau social, mais quand il s'agit de son PDG et fondateur, la donne est un peu différente. Dans un tweet, Jack Dorsey a pourtant annoncé fin novembre qu'il quittait ses fonctions pour se tourner vers de nouvelles aventures. Il cède sa place à Parag Agarwal, anciennement directeur technique de la société et qui se retrouve bombardé calife à la place du calife.



à la vitesse supérieure en décembre en ouvrant une action en justice visant à bloquer la transaction. Une nouvelle procédure de poids, qui vient s'ajouter aux initiatives des

régulateurs britanniques, chinois et européens, également peu enthousiastes à l'idée de voir Nvidia mettre la main sur la pépite britannique. Rendez-vous devant le juge en août 2022.

Rust devient le 2e langage officiel du noyau Linux



Eh oui ! Désormais, on peut contribuer en Rust. Des correctifs ont été déployés dans le noyau pour pouvoir supporter le langage. Ce support est désormais stable. Les drivers pourraient les premiers codes Rust. A vous de coder !

Drupal 8 est mort vive Drupal 7, 9 ou bientôt le 10

Drupal a rappelé que la version 8.x est désormais en fin de vie. Cela signifie que cette version n'est plus supportée. Pensez à migrer rapidement.

Pour rappel :

Drupal 7.x : fin de vie le 22 novembre 2022

Drupal 9.x : fin novembre 2023

Drupal 10 doit sortir mi-août 2022.



Felana Letrange

Je suis développeuse web JavaScript, avec un parcours très atypique.

Aime prendre des risques. Aujourd'hui heureuse maman et développeuse web. J'en veux toujours plus, ça ne sera jamais suffisant.

DE MAMAN AU FOYER À DÉVELOPPEUSE WEB : (UN) RETOUR DANS LA JUNGLE PROFESSIONNELLE

Quoi de plus flippant que d'avoir été en congé parental pendant 4 ans, et revenir sur le marché du travail ? On vous pose toujours cette même litanie : "Mais qu'avez-vous fait pendant cette période à trou sur votre CV?".

Avant d'être en congé parental, j'ai fait beaucoup de métiers, qu'on dit "alimentaires" (ça va de saisonnière dans un champ de pommes, à hôtesse de caisse en supermarché, je suis aussi passée par les centres d'appels et les parcs de loisirs). Il n'est déjà pas facile en soi de se remettre dans le contexte, le courage dont on a besoin pour passer la porte d'un premier entretien d'embauche après de longues années à s'occuper de ses enfants.

LE VIRAGE DE BORD

Alors, faire le choix de devenir développeur web en plus, c'est le parcours du combattant. Après mon congé parental, on m'a refusé plein de postes, on m'a fermé plein de portes. Je me suis donc tournée vers le freelancing, et là je tombe sur une boîte très bienveillante.

MAIS QU'A-T-IL BIEN PU SE PASSER ? L'enfance

Étant petite, j'étais déjà passionnée par les ordinateurs, les bons vieux ordinateurs Macintosh. Mais je n'ai jamais eu l'occasion d'en explorer les méandres. D'où je viens, il n'y avait pas tellement d'alternatives concernant les études informatiques, ou les aléas de la vie ont fait que je ne me suis pas penchée dessus.

La mauvaise rencontre

J'ai fait la rencontre de personnes toxiques également dans ma vie d'avant, mais le temps de m'extirper, et me voilà propulsée dans la recherche de "qui je suis ?", ou "qu'est-ce que je veux laisser dans ce monde comme empreinte ? »

ALORS COMMENT J'AI FAIT ?

Le plus difficile dans la vie, c'est le fait de se retrouver face à soi-même. C'est un très grand défi que je me suis imposée.

LA BELLE OPPORTUNITÉ QU'IL FALLAIT SAISIR

Après toutes ces portes qui m'étaient fermées, je me suis dit que j'allais me lancer free-lance, on m'a proposé une opportunité de pouvoir travailler à mon compte, en télétravail.

Cette occasion, je l'ai saisie, même si je n'avais pas encore les compétences ni les capacités requises pour ce qu'on m'a demandé de faire, mais je me suis lancée.

De là est née ma passion pour le développement web. Car j'ai travaillé pour une entreprise de création de magazines

numériques Madmagz, destinés aux établissements scolaires.

Ça correspond à mes valeurs, exercer un métier qui avait du sens, mais ça ne collait pas encore vraiment à ma passion, les ordinateurs, le monde du numérique.

Le fait de travailler entourée de développeurs web, de designer UX/UI a vraiment éveillé mes sens, et a contribué à ma décision de me lancer dans la reconversion professionnelle. Le créateur de cette boîte, ainsi que tous mes collaborateurs m'ont fait confiance, et c'est bien la première fois de ma vie qu'on me fasse autant confiance, surtout après des années à me faire rabaisser (13 ans durant).

J'avais des collègues qui ont su me déléguer des tâches, et je les ai acceptées avec beaucoup d'appréhensions, mais surtout avec beaucoup de fierté.

LE POINT DE NON-RETOUR

Me voilà donc partie à l'aventure de la reconversion professionnelle et devenir développeur web. Je cherche une école ou un centre de formation pour cela, mais encore une fois, je trouve porte fermée, avec des raisons aussi ubuesques les unes que les autres. Par exemple, " Vous n'avez pas les bases pour être développeur web", ou encore "vous n'avez pas un cursus scolaire adapté au monde de l'informatique".

L'ÉCOLE O'CLOCK

Je ne suis pas du genre à lâcher l'affaire aussi facilement, je me suis donc posée, et j'ai cherché l'école qui correspond le mieux à ma personnalité. Comme je suis quelqu'un de très autonome et autodidacte, la pédagogie de cette école me correspondait parfaitement. Les recherches ont porté leurs fruits, je suis tombée sur l'école O'clock, qui a su me donner ma chance. Je n'ai aucune notion ni connaissance du code, github, vscode, ma plus grande phobie au début de la formation était "le terminal de commande".

ET LA SUITE ?

Ceci dit, cela ne fut que mes débuts dans ce monde si vaste. Ce n'était que la porte d'entrée, car pour parfaire, ou compléter mes connaissances acquises durant ma formation initiale en développeur web, l'école o'clock a ouvert sa formation en alternance un bachelor Concepteur développeur d'applications. J'ai donc tenté cette approche de l'alternance.

Cependant, il faut savoir que la formation n'est qu'une porte d'entrée dans cet univers. Car on le sait tous, il en faut bien plus pour avoir le Graal.

Il y a le contexte, les besoins ou aspirations de chacun. Malgré ma motivation à continuer à apprendre, à évoluer tout en apprenant, mon contexte, je dirai handicapant, c'est mon âge.

Me voilà donc prise dans le tourment des candidatures sans réponses, ou encore les réponses ATS, bien que les entreprises aient mis tout en œuvre afin de les rendre aussi personnalisées qu'elles ne peuvent l'être ces pauvres ATS. Et ensuite, tu reçois 1, 2, 3, 4 réponses réelles, tu ne touches plus terre comme on dit. Une entreprise, des projets et des valeurs qui te parlent bien, qui rentrent bien dans tes critères, on discute, on passe les entretiens.

Arrive le moment fatidique de la question liée à l'âge, et pouf, tout s'envole, pensez bien qu'avec les aides de l'État, il est quelque part logique pour les entreprises de miser sur des profils de moins de 30 ans, et c'est compréhensible.

Bref, ceci n'est pas fait pour diaboliser le système, ni incriminer les entreprises, car il faut le dire, j'ai eu énormément de plaisir à discuter avec des recruteurs, des RH, un dirigeant d'entreprise, qui soit dit en passant est un très grand humaniste. Cela a été très instructif.

COMMENT OU QUOI FAIRE ?

J'ai listé mes critères professionnels :

- Stack technique
- Façon de travailler
- Taille de l'entreprise
- Secteur d'activité des entreprises auxquelles je veux postuler

J'ai listé ce qui me manquait pour me sentir légitime (bonjour le syndrome de l'imposteur) :

- La communication
- L'anglais professionnel à perfectionner
- Le front-end ; je me suis spécialisée en backend durant ma formation
- Et le dernier et non des moindres : LA PRATIQUE encore et encore.

Encore une fois, je ne suis pas ici pour faire peur, mais surtout pour éviter à celles et ceux qui souhaitent suivre cette voie. Il faut de la préparation avant, préparation de soi, préparation de son entourage proche, car eux aussi subissent ce changement radical.

Il faut de la préparation pendant, peu importe le choix du format de formation choisi. Il faut savoir que des hauts et des bas arrivent souvent. Pour ma part, je me posais la question suivante plusieurs fois dans une journée :

"Qu'est-ce que je fais ici et vais-je y arriver ?" Pas un seul jour ne passe sans que je doutasse, c'est là où la communication est très importante. Ce qui n'était pas mon fort au début. J'avais tort. Ne jamais rien lâcher par contre. Il faut aussi de la préparation pour l'après. Il faut être prêt

pour d'éventuels refus, et croyez-moi, il y en aura énormément, il y en aura plus que des entretiens. Ceci dit, tout cela a été très positif pour moi en tout cas. Pourquoi ? Parce que j'ai décidé d'en faire quelque chose de positif. Ça m'a permis de trouver ma propre stratégie de recherche, d'évolution.

J'ai compris également qu'il faut rester soi-même, sinon une reconversion professionnelle ne vaut pas le coup. Si on a décidé de changer de voie, de métier, c'est que ce qu'on avait avant ne nous convenait pas. Ou alors qu'on sait qu'on pouvait mieux faire, et faire les choses à notre image, en adéquation de nos propres valeurs.

En tant que développeuse web junior, on ne sait pas toujours par quel bout commencer. N'oublions pas, on ne peut pas tout savoir sur le web. Le monde du numérique est

en constante évolution, et à grande vitesse, que ce que tu as appris aujourd'hui, aura peut-être déjà changé demain.

Le plus important c'est de savoir trouver la solution adaptée à chacun, à chaque projet. Et si on ne sait pas, il faut aussi savoir dire qu'on ne sait pas. Car, de mon point de vue, une bonne développeuse sait le dire quand elle ne sait pas, mais qu'il va aller

chercher les solutions.

J'avais trouvé un stage début septembre, qui collait, mais pas à 100% avec mes 2 listes. Mais c'est normal, j'ai fait mes listes après ce stage. Pourquoi j'ai arrêté ? Parce que je n'ai pas encore su gérer le stress, c'est la première fois que je faisais un stage dans ma vie, oui tout ne fait que commencer pour moi. Le projet était innovant, très enrichissant, les technos utilisées m'intéressaient. Seulement voilà, c'étaient des technos que je n'avais jamais vu avant, sachant que je suis junior. J'ai donc fait le choix de rétro pédaler, pour mieux rebondir. Je n'ai pas manqué d'en parler clairement et franchement avec mon maître de stage. Cela m'a aidé à faire ma liste, à faire mes choix, et à filtrer ce que je voulais.

CONCLUSION

Aujourd'hui, quand je regarde "au rétroviseur", je vois le chemin parcouru et je suis fière. Je n'ai pas encore trouvé de boulot. Cependant, j'ai réussi à trouver un stage, qui me sera très formateur. Durant ce stage, je vais développer de nouvelles fonctionnalités, apprendre de nouvelles technos qui attisent ma curiosité depuis plusieurs mois. Je vais également continuer à approfondir ce que je sais déjà, compléter ce qui me manque, c'est-à-dire, apprendre le côté front-end, perfectionner mon anglais, tout cela par la pratique.

Ne serait-ce pas merveilleux ? OUI, et par-dessus le marché, je travaillerai en full remote. Je serai dans une entreprise qui privilégie l'humain avant tout, qui met un point d'honneur à la vraie signification de la reconversion professionnelle, une entreprise qui voit au-delà d'un simple CV. Et n'oubliez pas, il n'y a pas d'âge pour tout changer, pour tout commencer ou tout recommencer. Tout est possible.



Romy Alula

Romy est consultante en développement Web chez Codeworks. Membre active de Ladies of Code et Duchess France, elle y accompagne de jeunes dev en tant que marraine. Sur des sujets comme le Craft, la préparation d'entretien technique ou encore la négociation. Depuis mars 2021, elle anime le hands-on sur le TDD, chez les Ladies of code. Guidée par sa passion et sa curiosité, elle partage ses apprentissages et découvertes avec d'autres curieuses et curieux au travers de coding dojos, meetups et autres conférences. @Goumies

Reconversion réussie et carrière

Il me tient à cœur de partager mon retour d'expérience de la reconversion professionnelle au métier de Développeuse Web. Nous sommes de plus en plus nombreuses à avoir un parcours ou profil dit "atypique". En terminale, j'ai découvert les algorithmes et Alg'exec. Pour moi, c'était magnifique d'analyser un problème et de faire exécuter mon algorithme par un ordinateur. Avec deux parents ingénieurs en Informatique et un bac spécialisé en Informatique et Gestion, j'ai longtemps cru que l'Informatique ce n'était pas pour moi. Peur de m'ennuyer, peur que ce soit trop sérieux. J'ai toujours aimé concrétiser mes idées via l'outil informatique (Creative Suite, 3ds max...), mais pendant longtemps, je refusais de m'orienter vers le code et l'informatique en général. J'ai presque 10 ans d'expérience en tant que Créative dans l'audiovisuel et la Communication. J'ai été Cheffe de Projet multimédia. Mais ça, ce n'était vraiment pas pour moi.

Tout a commencé avec le Web

Je savais que j'avais déjà un attrait pour le code quand je posais des questions à mes parents pendant qu'ils travaillaient. Je m'intéressais déjà au Web, à l'époque. Au cours de mon cursus d'infographiste 3d / Graphiste, le besoin de présenter mon travail en ligne m'a poussée à m'intéresser d'abord à l'ActionScript. Plus tard, à l'HTML5 et au CSS3. Je voulais créer mon propre site en partant d'une feuille blanche. Je gardais un souvenir marquant de mon père qui développait des sites qu'il visitait avec Netscape. Et ça, sur son temps libre, par plaisir.

En 2011, j'ai repris le chemin de l'école pour obtenir un Master 1 en Gestion de projet et Marketing interactif. Deux intervenants ont tout de suite retenu mon attention. Ils avaient un profil que je découvrais, celui de Creative Technologist. Sortie d'un cursus en Jeu Vidéo Animation, sans valider le diplôme, j'avais des compétences en création numérique. La liberté de créer de toutes pièces un projet de l'idée à la livraison a vraiment décuplé mon envie de coder. Merci à Vincent et Arnaud, mes profs de Sup'Career !

Je ne comprenais pas l'enthousiasme d'Arnaud, Directeur Technique et Développeur, devant mes compétences techniques et mon profil créatif. C'est avec beaucoup d'intérêt que je découvrais, grâce à lui, l'existence du HTML5 et du CSS3. J'ai approfondi mes connaissances des technologies standard du Web grâce aux MOOC (Massive Open Online Course).

Le courage de se lancer

Réellement se lancer dans une reconversion demande beaucoup de courage. D'acceptation de se mettre volontairement dans une position vulnérable. Les difficultés que je vivais, en tant que Créative professionnelle : peu ou pas d'entretiens d'embauche ; quand je trouvais une mission en free-lance, il fallait relancer et insister pour être payée ; les prospects négociaient toujours plus à la baisse. L'absence de stimulation ou motivation due à des projets qui n'étaient plus qu'alimentaires et mon incapacité à

dessiner plus pour progresser et créer mon style, tout ça m'a secouée. J'ai dû faire preuve de résilience.

C'est bien plus tard, en 2017-2018, que j'ai concrétisé tout ça avec mon titre professionnel de Conceptrice Développeuse Informatique. En tant que demandeuse d'emploi, j'ai pu m'inscrire via Pôle Emploi. Il a fallu beaucoup insister pour ne serait-ce que m'inscrire à la session de recrutement qui m'intéressait. Il y a des places réservées aux demandeurs d'emploi. Le Conseil Régional finance la formation et rémunère les stagiaires de la formation professionnelle. Pour mettre toutes mes chances de mon côté, je reprenais le contenu de la formation et les exercices pratiqués dans la journée, à la bibliothèque de Beaubourg (parce qu'elle ferme tard). Je le faisais uniquement parce que j'avais décidé que cette formation à l'AFPA serait la dernière que je m'autoriserais. J'étais renforcée par le soutien de ma famille.

Avant d'intégrer ma formation, j'étais mère au foyer depuis 2014. J'avais pu travailler sur de rares missions en free-lance en maquettage Web essentiellement.

On s'accroche

Cette formation a été un véritable chamboulement. Neuf mois très intenses ! Les changements impliqués étaient surtout d'ordre logistique et organisationnel. Avec deux enfants en bas âge, il a fallu courir et prendre du temps pour assimiler beaucoup d'informations. Je travaillais le soir dans les bibliothèques ou les pubs et autres quand je finissais vraiment tard. C'était difficile, mais ça valait le coup. Comme il m'était impossible de travailler à la maison avec les enfants, je maximisais mes soirées studieuses et mes journées au centre de formation.

Au niveau du contenu de la formation, quand j'avais du mal à comprendre, je pouvais faire appel à ma mère et un de mes oncles pour m'expliquer les concepts clés. Le suivi individualisé des formateurs a aussi été d'une grande aide.

Ce qui m'a aussi boostée, c'est le travail en équipe pendant les 4 projets "fil rouge". 6 semaines, 1 cahier des charges, 4 développeurs et 3 formateurs pour répondre aux questions Métier.

En plus de ça, j'ai rencontré un groupe de personnes motivées, intéressantes, drôles et encourageantes. Notre groupe "Feng Shui", avec notamment, Samira, une autre jeune mère. Nous nous sommes accrochées et avons validé notre titre professionnel.

En tant que femmes, jeunes mères de filles et 2 seules femmes à avoir tenu jusqu'au bout, il était important de réussir.

Une fois mon titre professionnel obtenu, j'ai choisi de travailler dans les ESN. Ce choix était motivé par la volonté de travailler dans une variété de contextes et projets.

Le plaisir d'apprendre

Avec mon passage par l'AFPA, j'ai appris à apprendre. Chose capitale dans notre métier. Les technos évoluent vite. Les fondamentaux restent. Aussi, il arrive souvent qu'il faille désapprendre pour réapprendre. Je me rappelle, notamment, quand j'ai dû passer d'Angular.js à Angular 6. Il s'agit d'un tout autre paradigme. Au-delà d'être opérationnelle, je me suis attelée à me former sur les concepts de base du framework. Dans le cabinet de consulting que je venais de rejoindre, l'expert JavaScript m'avait donné des pistes de réflexion. "On n'est pas à l'école ici !" : ce sont les mots de mon employeur qui m'ont confirmé le fait que, pour progresser, je devais partir.

L'amélioration continue : gage de progression

Dans le processus d'amélioration continue, je trouve le même plaisir à apprendre un nouveau morceau de piano. Ou encore la redécouverte d'un morceau joué par cœur, à la reprise, à l'âge adulte, de cet instrument que j'affectionne. C'est une affaire de méthode. Pour un nouveau morceau, je lis la partition, main droite, puis main gauche. Je repère les positions des doigts indiquées. Ensuite je répète une séquence en boucle, main droite puis main gauche. Une fois que le jeu est fluide, je joue la séquence avec les deux mains. À mesure que j'avance dans l'assimilation du morceau, je peux y ajouter l'interprétation.

À l'arrivée en mission client, la prise en main du projet suit plusieurs phases :

- La préparation de l'entrée en régie consiste à reprendre mes notes rédigées pendant l'entretien. Lire un article ou deux pertinent(s) dans le contexte que je vais rejoindre. Par exemple, sur le Typescript avancé.
- Le premier mois, c'est beaucoup d'observation et de pair-programming. Je suis opérationnelle sur de "petites fonctionnalités".
- Dès le mois suivant, je suis autonome sur des sujets plus importants.

- Pour suggérer des améliorations structurantes, je vais faire un POC (Proof Of Concept). C'est un projet isolé qui va venir illustrer la techno, la méthodologie que j'aimerais apporter au projet. C'est une bonne base pour échanger avec le reste de l'équipe.
- En fonction des besoins, je monte en compétence sur des sujets spécifiques. Grâce à des plateformes comme Pluralsight, solution choisie au sein de CodeWorks. Autrement, je peux aussi faire appel aux communautés CodeWorks. Tout comme je peux partager ma problématique avec les participants des coding dojos et autres forums ouverts auxquels je participe.

« Elle » parmi « ils »

En tant que profil reconverti, sachez que vous rencontrerez parfois des personnes, au mieux sceptiques, au pire défiantes ou méprisantes. Si cette personne vous malmène en entretien technique, qu'est-ce qu'il en sera, une fois en poste et surtout en période d'essai ? Mon expérience de ce genre de rencontres m'a appris à savoir refuser de rejoindre une entreprise ou accepter l'échec de l'entretien pour un meilleur environnement. La période d'essai, que vous soyez reconverti.e ou non, est un essai pour les deux parties. L'entreprise a besoin de votre travail, au même titre que vous avez besoin de travailler.

Mentalement, moralement, ça a été dur. Une fois qu'on rejoint un contexte professionnel sain, on prend alors conscience qu'on a trouvé sa place. Et là, on apprécie tout le chemin parcouru, les difficultés surmontées.

Comment trouver sa place dans des équipes entièrement masculines ? Là où il était monnaie courante de me couper la parole, en réunion, ou d'accepter mes propositions quand elles étaient reformulées par un collègue, j'ai bien choisi ma mission actuelle à la Société Générale. Cette fois, notre équipe fonctionne dans l'échange. Je me suis intégrée grâce à ma curiosité naturelle. En entretien déjà, le courant passait très bien et on s'est mutuellement appris des choses, au sujet de TypeScript. Le pair-programming systématique avec chaque membre de l'équipe, pendant mon onboarding, a beaucoup contribué à cette intégration. C'est typiquement le genre de pratiques sur lesquelles je pose des questions en entretien de mission. Quand elles sont en place et abordées par la positive, c'est un bon indicateur d'ouverture d'esprit.

Il est temps de bouger : mon expérience et mon instinct en guise de boussole. Avec le temps, j'ai compris que je devais me fier beaucoup plus à mon instinct. Je fais le point pour m'assurer que je suis bien là où je veux être. Si la liste des compromis est plus longue que celle des points positifs, il est temps de partir pour de meilleures aventures, toujours dans un risque calculé.

La reconversion : outil d'autonomisation

À l'heure où j'écris cet article, je suis consultante chez CodeWorks (<https://www.codeworks.fr>). Ce matin même, je suis à nouveau intervenue au sein du programme Horizon

Numérique de Social Builder (<https://socialbuilder.org/>). Dans une présentation plus détaillée, je partage mon parcours de reconversion avec un public de demandeuses d'emploi. Il existe de nombreuses initiatives du même genre, notamment L'Digital (<https://www.ldigital.org>) qui propose un accompagnement dans les métiers du numérique, des ateliers en milieu scolaire.

Participer à des meetups et conférences autour du Craft m'a permis de rencontrer des personnes passionnées, ouvertes d'esprit. Celui que je considère comme mon mentor m'a présenté à ma marraine chez les Duchess. Merci à vous, Pauline et Yvan. Vous avez joué un rôle déterminant dans le succès de ma reconversion ! Vu la charge de travail et le temps que demande ce genre de projet, il est important de s'entourer de positivité. Ma formation à l'AFPA n'aurait pas été la même sans les "feng shui". Notre groupe s'est accroché et soutenu pour aller jusqu'au bout et obtenir le titre.

Partage et transmission : tous gagnants

Dans le partage et la transmission de savoir, les deux parties apprennent. Comme dit Kyle Simpson : "Si vous attendez de "maîtriser" une compétence, avant de partager ce que vous avez appris avec les autres (blogs, vidéos, talks...), vous avez attendu trop longtemps. Et le reste d'entre nous passons à côté du plus important : votre parcours d'apprentissage". C'est avec beaucoup de plaisir que j'ai participé à des coding dojos et autres sessions de katas. La Global Day Code Retreat fait partie de ces événements récurrents où on peut apprendre beaucoup, dans une ambiance ludique.

Mentorer des devs ou aspirantes devs me permet de prendre du recul sur ces 4 dernières années. Chacune d'elles, forte de sa détermination et sa passion, m'apprend aussi en retour. Claire vient de finir son alternance dans une startup au produit intéressant. Elle m'a fait découvrir React Admin, un framework dédié au développement de back office. Ou encore TypeORM, un ORM orienté ESNext (ES6 et plus) et Typescript.

J'ai moi aussi fait appel à des mentores. Aude m'a permis de m'entraîner à l'Event-sourcing et au CQRS en javascript. Animer le hands-on TDD m'apporte aussi beaucoup. Je redécouvre des katas classiques, sous un nouveau jour, avec des approches et solutions originales. L'enthousiasme des participantes, qui souvent découvrent le TDD, en plus du mob-programming, me permet d'améliorer le hands-on, à chaque édition. Merci à Hadrien et Thomas, les organisateurs du Mob programming Paris, pour leurs conseils et ressources.

Le temps dédié à la veille est non négociable

Comme dit plus haut, les technologies évoluent vite. Pour rester à jour et assurer son employabilité, la veille technologique est indispensable. Que l'on soit salarié chez un client final ou prestataire. C'est pourquoi le temps dédié à la veille est, pour moi, non négociable. Nous avons tous et toutes les mêmes 24 heures. Pouvoir mener une veille technologique sur son temps de travail est un vrai plus pour l'entreprise.

Je demande systématiquement, en entretien, si un temps collectif est dédié à la veille. Dans les entreprises où missions que je veux rejoindre, il y a soit une journée mensuelle ou une après-midi hebdomadaire. Et ce parfois en plus de guildes (front, back, cloud, Craft). Chez Fnac Darty, des Communities Of Practice s'organisaient, en toute autonomie. À l'époque où j'y étais en mission, il y avait même un projet de club de lecture. C'est typiquement le genre d'environnement où j'ai pu partager des connaissances et retours d'expériences avec des devs actifs et enthousiastes.

Parlons salaire : le silence profite aux employeurs

Bien que le salaire reste encore tabou, parler salaire avec mes collègues a été révélateur, à chaque fois. Que ce soit avec ma copine Samira, membre des FengShui qui m'a mise en garde, avant la sortie de l'AFPA, sur un seuil minimal à partir duquel elle estimait qu'une offre était acceptable pour mon profil et mon parcours. Ou avec mes camarades des KataCombe, promo la plus expérimentée de la Combe Du Lion Vert (<https://la-combe-du-lion-vert.fr>). Ou encore avec l'un de mes collègues de mission, lui aussi passé par une reconversion. Encore une fois, il s'agit d'échanger avec des personnes de confiance. Ça vous permettra de négocier en connaissance de cause, sans mettre qui que ce soit en porte-à-faux, bien évidemment. Après, certaines entreprises sont transparentes avec leur grille de salaires et augmentations.

MES CONSEILS

À toutes personnes en reconversion ou qui aimeraient se lancer :

- partez de ce que vous, vous voulez, de vos compétences. De l'effort que vous êtes prêt.e à fournir, tout en prenant soin de vous, pour pouvoir durer.
- renseignez-vous un maximum.
- rencontrez des professionnels, dans le cadre de meetups, conférences ou encore via LinkedIn. Et posez-leur des questions.
- à la sortie de formation, nombreux sont ceux qui ont du mal à trouver leur premier emploi. Nombreux sont ceux qui finissent par abandonner. Même face au manque de temps et aux refus, persévérez pour vous faire l'expérience qui sera la vôtre. Gardez votre objectif principal en tête. Croyez en vous, vos capacités et votre aptitude à rebondir. Rappelez-vous que vous venez de valider votre parcours de reconversion. Un seul "oui" suffit.
- la technique tient une place importante. Mais elle ne fait pas tout. Votre créativité et toutes vos expériences passées vous permettront d'apporter un regard neuf.
- construisez-vous de solides fondamentaux avec les concepts de base. Et pratiquez. Les coding dojos et autres Global Code Day Retreat sont très bons pour ça.
- la curiosité est un atout majeur. Gardez toujours un état d'esprit d'étudiant.e. Il y a toujours quelque chose à découvrir / apprendre / approfondir.
- le meilleur pour la fin : entourez-vous de personnes qui partagent les mêmes passions et valeurs que vous. Éloignez-vous de la négativité, préservez votre énergie et votre temps.

Quand développer (re)devient une évidence

Selon une étude publiée en 2018 par l'OPIIEC (Observatoire Paritaire des Métiers du Numérique, de l'Ingénierie, des Études et du Conseil et des métiers de l'évènement), le secteur du numérique est constitué à 74% d'hommes **[Figure 1]**, ce qui en fait « Le secteur le plus masculinisé de la branche [des Métiers du Numérique, de l'Ingénierie, des Études et du Conseil et des métiers de l'évènement] » [1].

Il est même courant d'entendre que « Les femmes sont moins présentes que les hommes dans l'IT, surtout lorsque cela nécessite des compétences techniques ». Pourtant, m'éloigner du « code » a été pour moi un tremplin qui m'a permis de m'épanouir dans de nouvelles compétences techniques. Après 3 ans dans une équipe de développement, je n'étais plus inspirée par mon métier. Il fallait, sans maîtriser les technologies (à mon goût au moins) coder vite (toujours plus vite) des fonctionnalités (ou des technologies) dont je ne voyais pas toujours l'intérêt. J'ai intégré une équipe de tests fonctionnels et manuels. J'étais ravie : « La technique ce n'est pas ce qui m'intéresse », disais-je naïvement.

L'évolution du métier de testeur·euse

À cette époque, si vous aviez demandé à certain·e·s une définition du poste de testeur·euse, cela aurait pu donner : « Cliquer sur des boutons en suivant des scénarios prédéfinis ». L'agilité, l'intégration continue, le devops ont fait évoluer les pratiques du domaine des tests logiciels. Ainsi, dans un nombre croissant de structures, afin de garantir un rythme plus rapide de livraison en s'assurant de la fiabilité du produit, l'automatisation des tests, notamment de non-régression, prend désormais une place importante. Les tâches possibles d'un·e testeur·euse sont nombreuses. Mes activités de testing vont de la diffusion des bonnes pratiques des tests au support fonctionnel en passant par la conception et l'exécution des tests manuels ou automatiques **[Figure 2]**.

Vous remarquerez que quelques années après ma spécialisation dans le domaine du test et de la qualité logiciel, j'implémente donc des tests automatiques. Pour automatiser les tests, une grande variété d'outils est disponible sur le marché. Pour les tests des API, j'utilise Postman [2][3] et pour les tests end-to-end Cypress [4][5]. Ces deux outils permettent d'écrire des tests dans le langage JavaScript que je n'avais pas pratiqué dans mon passé, je l'ai donc appris, mis en pratique puis j'ai utilisé, de nouveau, l'outil de gestion de versions décentralisé Git. Ensuite, pour améliorer l'existant, j'ai cherché à aller plus loin, j'ai modifié les pipelines d'intégration continue, utilisé docker pour exécuter les applications, fait de la veille pour



Figure 1 : Proportion des hommes dans le secteur du numérique

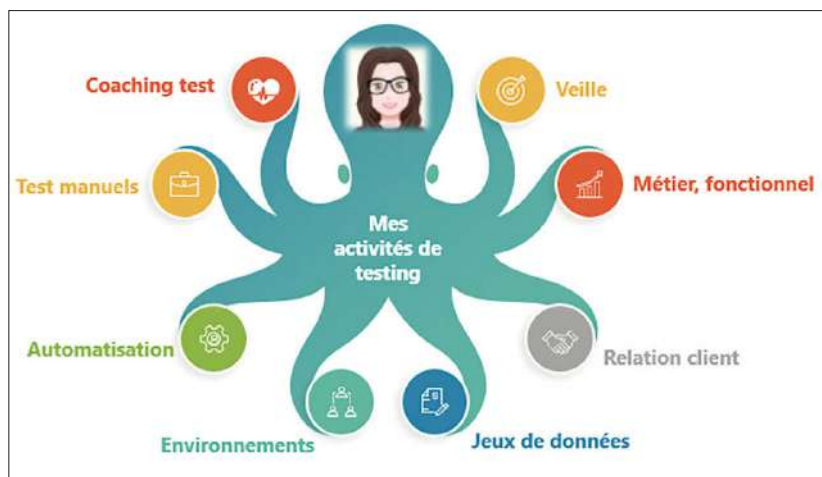


Figure 2 : Mes principales activités de testing en 2021

trouver quelles autres bibliothèques ou outils pourraient avoir de l'intérêt. Par exemple, j'ai récemment commencé à apprendre l'utilisation de PACT (test de contrat d'interface) [6].

La formation est un concept clé, n'hésitez pas à suivre des cours en ligne. J'apprécie tout particulièrement la Test Automation University [7] qui propose des cours, en anglais, complètement gratuits comportant un questionnaire permettant de valider les acquis. Vous pouvez aussi participer à des *challenges*. Le Ministry of Testing en propose plusieurs à réaliser sur 30 jours [8], les « 30 days of Agile Testing », par exemple, m'ont permis de me remettre en question sur plusieurs points : en le suivant j'ai pratiqué le pair-programming et effectué des revues de codes. Postman propose les 30 jours de Postman pour les développeur·euse·s [9], plus technique que les liens cités plus hauts, avec quelques recherches sur le net l'obtention du badge est accessible. Évidemment, la qualité ne s'arrête pas à l'automatisation des tests, la pratique des revues de code, le pair programming, car toutes les (bonnes) pratiques de développement me sont ouvertes. Je peux



Anne-Laure Gaillard

Spécialisée dans la qualité logicielle, passionnée par le test logiciel et l'agilité, partisane de la doctrine « La qualité est l'affaire de tous ».

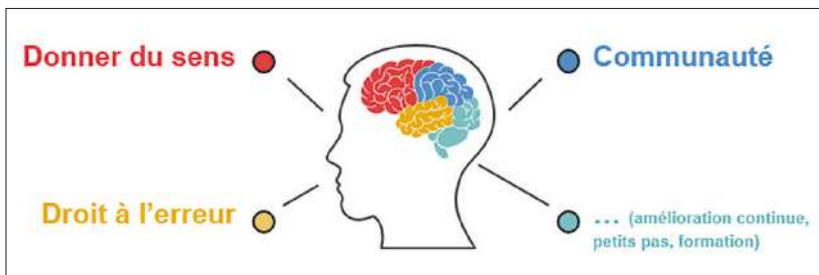


Figure 3 : Des moteurs que j'ai identifiés



Figure 4 : Citations de femmes célèbres

désormais me qualifier de plus technique que lorsque j'étais développeuse.

Programmer, c'est cool

Je suis revenue au développement parce que le domaine du test a évolué. La question importante est de savoir pourquoi est-ce que je me plais à faire ces tâches techniques alors que je pensais vouloir m'en éloigner ?

La réponse est multifactorielle. Tout d'abord, je trouve un réel intérêt pour le sujet. Au départ parce qu'il m'évitait de répéter des tâches, maintenant car il apporte une plus-value à toute l'équipe tel qu'un feedback plus rapide lors des modifications du logiciel et une confiance augmentée lors des livraisons. Ensuite, lors de mon démarrage, j'ai été la

première à mettre en place des tests automatiques dans mon équipe. En dehors de mon entreprise, la maîtrise des logiciels et techniques d'automatisation des tests était relativement récente, je n'ai donc pas eu le syndrome de l'imposteur. J'ai alors compris que je pouvais apprendre, échouer, recommencer et m'améliorer.

Même si la culture est en train de changer en France, certaines équipes ont du mal à accepter la culture de l'échec, vous pouvez tout de même l'appliquer à votre échelle et avancer par petits pas.

Enfin, j'ai trouvé de nombreuses communautés très soudées et non jugeant avec lesquelles nous partageons continuellement. La communauté de testeurs <https://www.ministryoftesting.com> propose, par exemple, un forum et un espace de travail Slack permettant aux débutants et expérimentés d'échanger. De nombreuses autres communautés existent, qu'elles soient locales ou plus spécifiques comme celles des utilisateurs francophones de Cypress ou encore sur des sujets connexes à cet article comme l'Agilité ou les Software crafters.

Pour terminer cette partie sur mon parcours, j'ai appris que j'étais plus épanouie et performante en suivant mes centres d'intérêt, en partageant avec mes pairs et me donnant le droit d'échouer [Figure 3].

Et vous ?

J'ai pu travailler avec de nombreuses femmes dans le domaine du test logiciel avec tellement d'histoires différentes : des jeunes diplômées en UX, des expertes métier ou en réorientation professionnelle, des consultantes, des développeuses et même des retraitées en portage salarial. Vous vous reconnaissez dans ces parcours ? Vous voulez, comme moi (re)partir vers le développement et faire évoluer les statistiques ? N'hésitez plus [Figure 4] et rendez-vous dans une communauté pour en apprendre davantage telle que la communauté de promotion des femmes dans les métiers de la technologie et du numérique <https://www.duchess-france.org>.

[1] https://www.opiiec.fr/sites/default/files/inline-files/Portrait_Statistique_Branche_RAPPORT_NATIONAL.pdf

[2] <https://www.postman.com/>

[3] Un cours, en anglais, pour s'initier aux tests d'API et à Postman <https://testautomationu.applitools.com/exploring-service-apis-through-test-automation/>

[4] <https://www.cypress.io/>

[5] Un tutoriel d'initialisation, en anglais sur Cypress <https://testautomationu.applitools.com/cypress-tutorial/>

[6] Le cours sur PACT <https://testautomationu.applitools.com/pact-contract-tests/>

[7] <https://testautomationu.applitools.com/>

[8] Liste des challenges 30 jours du ministry of testing, en anglais, <https://www.ministryoftesting.com/dojo/series/30-days-of-testing>

[9] Challenges Postman, en anglais <https://blog.postman.com/public-workspace-for-api-trainings-postman/>

Si tu n'es pas passionnée, tu n'es pas une bonne développeuse ?

Nous connaissons tous dans notre entourage professionnel un développeur passionné par ce métier. On le reconnaît au fait qu'il passe des heures à coder par exemple. En véritable passionné, il se documente, consacre énormément de son temps libre à coder et ne parle que de code pour grossir le trait.

Aussi passionnant qu'il puisse être, si on ne ressent ni n'exprime cette passion pour le code, il peut vite survenir la sensation d'être un imposteur, une fraude face à ce passionné. On peut vite être amené - si finalement on ne ressent pas cette passion - à se poser des questions sur ses compétences professionnelles, car le passionné passe plusieurs heures sur son temps libre sur des projets personnels là où on n'en a ni l'envie, ni le temps, ni même le besoin.

Mais alors faut-il vraiment être passionné pour exercer le métier de développeur web, voire la passion est-elle une prérogative au sentiment de légitimité ? Qu'est-ce que la passion peut bien apporter ? L'injonction de la passion ne serait-elle pas en fin de compte un piège ? Si ce n'est pas la passion, qu'est-ce donc ? Enfin, la passion est-elle LE logiciel indispensable à quiconque souhaitant coder ?

Sur les traces de la passion

Qu'est-ce que la passion ?

La passion est définie[1] au sens littéral comme étant un état affectif intense et irraisonné qui domine quelqu'un ou comme un penchant vif et persistant. Appliquée au monde du travail, une personne passionnée par son travail ne perçoit pas les tâches liées à sa fonction comme un sacerdoce.

Comme le pointe justement Philippe Laurent[2] :

« L'homme passionné par son travail n'a pas le sentiment de travailler [...] Il fait ce qu'il aime sans se forcer ni se raisonner à aimer ce qu'il fait ». Cette vision de la personne passionnée par son travail est un objectif que l'on rêverait tous plus ou moins d'atteindre, d'autant plus lorsque l'on exerce un métier dans le champ de la créativité.

Il est vrai qu'être développeur, c'est faire preuve de créativité, car nous construisons au quotidien des applications/sites amenés à être utilisés dans un certain contexte. Être développeur.se, c'est donner vie à une idée à l'aide d'outils techniques, de langages permettant d'interagir avec la machine en vue de la matérialiser. C'est un métier technique, car nous utilisons des solutions dites technologiques et manuelles pour parvenir au résultat escompté.

Certaines tâches peuvent s'avérer rébarbatives et fastidieuses, la correction de bugs en est un exemple. La passion peut alors faciliter la réalisation de ces tâches paraissant moins pénibles pour la personne passionnée. La passion apparaît donc comme la clé du bonheur au travail. Elle apporterait des bénéfices certains, mais qu'en est-il réellement de la passion au travail ?

La passion au travail

Selon le modèle dualiste de la passion[3] en psychologie, les personnes s'engagent dans des activités qui se démarquent du fait de leur caractère « agréable, important et parce qu'elles s'arriment particulièrement avec l'identité de l'individu ». Ces activités sont intériorisées par les individus dans le soi et l'identité. Le processus d'intériorisation peut se faire de façon contrôlée ou autonome ce qui influe sur le type de passion développée. Il est alors fait la distinction entre deux types de passion : la passion harmonieuse et la passion obsessionnelle.

La passion obsessionnelle émane d'une intériorisation contrôlée de l'activité dans l'identité et le soi. « Il peut s'agir du sentiment d'excitation incontrôlable procuré par l'activité ou encore de contingences rattachées à l'activité ». Le passionné obsessif éprouve un besoin incontrôlable d'exercer l'activité qu'il aime et le fait de manière rigide. Évidemment, être un passionné obsessionnel peut conduire à une meilleure performance, toutefois, la passion obsessionnelle entraîne des conflits entre les différentes sphères de la vie du passionné. Une développeuse qui délaisserait sa vie sociale et familiale, car obnubilée à l'idée de terminer un projet par exemple, avec ou sans contrainte de temps. A contrario, la passion harmonieuse résulte d'une intériorisation autonome de l'activité dans l'identité et le soi de l'individu. Cette sorte de passion autorise l'individu à choisir librement d'accepter l'activité comme importante et comme faisant partie de son identité sans aucune contingence. Dans ce cas de figure, une harmonisation est possible entre l'activité passionnante et les autres sphères de la vie. De cette harmonisation, découle des conséquences positives pour l'individu, « sans ruminer ou être frustré de ne pouvoir exercer son activité passionnante ». La développeuse passionnée, ici, pourra développer plusieurs passions en dehors du code et remettra à plus tard les projets non urgents pour profiter d'une soirée entre amis.

En opposant passion harmonieuse et passion obsessionnelle dans un contexte professionnel, nous réalisons que lorsque l'une est saine, l'autre est porteuse de conséquences.

Existe-t-il des pièges liés à cet état de passion ?

Quels sont-ils ?

Le piège de la passion

Être passionnée par son métier paraît idyllique. Le travail semble moins pénible. Dans le fond, pourquoi on ne se lancerait pas tous dans cette quête de la passion ? Les



Helvira Goma

Développeuse web - essentiellement dresseuse d'éléphants. Lorsqu'elle n'écrit pas des lignes de code, elle prête sa voix et sa plume à son podcast *Resilient Evil* où elle y raconte des histoires autour de la résilience. C'est avec l'envie de montrer aux femmes que le développement est une voie professionnelle envisageable qu'elle crée *Motiv'Her* en 2021.

choses n'en seraient que plus simples, non ? C'est là qu'il semble important d'équilibrer la balance, car oui, être passionnée comporte son lot d'inconvénients. Selon une étude[4] [5], les gens considèrent qu'il est plus acceptable que des employés passionnés par leur job travaillent plus et gratuitement que des employés lambda. D'autres en arrivent à affirmer qu'il est légitime de demander aux personnes passionnées par leur job de prendre sur leur temps personnel et familial pour travailler le week-end.

C'est en cela que réside le piège, la passion serait une sorte de cadeau empoisonné. En effet, avoir la sensation que la tâche est moins ingrate ne la rend pas pour autant moins ingrate. Cela n'occulte pas la réelle pénibilité. Raison pour laquelle, prises dans ce piège, le burn-out est d'autant plus probable pour les personnes passionnées. Ce qui crée de la confusion, car comment peut-on faire un burn-out alors qu'on exerce un métier qui nous passionne ? La passion forme une bulle autour de soi de laquelle il faut pouvoir sortir pour en éviter les pièges dans le monde du travail notamment.

Il est clair que la passion ne fait pas tout dans le bonheur au travail ou bien dans le sentiment de se sentir développeur. Se posent alors les questions : si on n'est pas passionnée et si être passionnée comporte autant de contre que de pour, que faire ? Faut-il être passionnée pour être une bonne développeuse ?

S'affranchir du dictat de la passion Le développeur se doit-il d'être passionné ?

C'est un postulat qui circule de manière récurrente au sein des développeurs : le développeur doit être passionné s'il veut mener une carrière « digne de ce nom ». Tu n'es pas un développeur respectable si tu ne passes pas tout ton temps libre à faire des exercices d'algorithme ou à créer des applications / sites. La passion est érigée en totem à absolument détenir pour ne serait ce qu'être légitime. Ce qui en pratique est totalement faux.

Cette tendance visant à penser que seule la passion est un gage de réussite et d'excellence en tant que développeuse est inutile si ce n'est dangereuse. En effet, elle induit que le développeur qui ne ressent pas cet amour pour ce métier n'en est pas digne et donc nie probablement son expérience. Car que vaudrait-elle si elle n'était pas teintée de passion ?

De là naissent une forme d'élitisme ainsi qu'une certaine souffrance. Tel que le signale à juste titre Medhi Zed[6] : « Les passionnés n'ont aucun talent inné. Leur maîtrise s'est faite avec le temps et ils ont tendance à l'oublier. Jamais une compétence n'est devenue autant un mythe dans la tête des gens ».

La passion n'est donc pas un super pouvoir à acquérir sous peine de ne pas être une bonne développeuse. On est alors en droit de se demander ce qui fait qu'on puisse être performante en tant que développeuse sans pour autant posséder le « feu sacré » ?

Être passionnée, c'est être une bonne développeuse ?

Certains expliquent cette injonction à la passion par le fait que le métier de développeuse est un métier complexe et sans passion, il semble difficile pour un non passionné de venir à bout des différentes difficultés que l'on rencontre. En réalité, il est possible de distinguer trois catégories de développeurs : le développeur passionné évidemment qui vit, respire et parle code tout le temps ; le développeur qui a un intérêt relativement modéré et qui développe sans grande effusion d'émotions comme il est commun dans d'autres secteurs d'activité ; le développeur pour qui le développement est juste une question d'argent et pour qui les journées se résument à attendre que cela passe en assemblant les bouts de code ici et là. Or, il peut s'avérer que l'on compte parmi les meilleures, les développeuses appartenant à la seconde catégorie. Pourquoi ?

La passion naît avant tout de la pratique. On ne se réveille pas un matin mordu de code. C'est en essayant de comprendre la logique derrière les langages informatiques, en étant curieux de ce qui se passe (au cours de sa veille technologique) et en pratiquant que son intérêt pour le métier grandit.

Et il n'y a aucun mal à ne pas être passionnée par le code. Aucun mal à ne pas consacrer ses soirées libres et ses week-ends à un projet personnel. D'autant plus que nous avons tous (passionné ou non) une flopée de projets personnels que l'on n'a jamais terminés ni mis en ligne. Certes, il y a des impératifs dans le métier de développeur tels que faire de la veille technologique. Mais, cela n'implique pas d'y consacrer toute sa vie. Le tout étant une question d'organisation.

La clé d'une carrière réussie - tel qu'on le souhaite - dans le développement est le travail quoiqu'on en dise. Rien que le travail. La passion n'est pas une obligation, ni un passe-partout et encore moins une compétence. Peut-on alors trouver un compromis ? Une autre direction que celle de la passion ?

À la découverte du plaisir

Le Larousse définit le plaisir[7] comme étant un état de contentement que crée chez quelqu'un la satisfaction d'une tendance, d'un besoin, d'un désir. Il serait un excellent compromis à la passion. Il obéit à un paramètre moins suggestif qu'est la satisfaction.

En tant que développeuse, nous passons par toutes les couleurs de l'arc-en-ciel. Notre quotidien est ponctué de colère lorsqu'on tombe sur un bogue récalcitrant ; de tristesse / nostalgie lorsqu'un projet se termine et de joie quand on trouve une solution à une problématique. C'est ainsi que la notion de plaisir pointe le plus souvent le bout de son nez. Certes, le rouge nous vient lorsque nous tombons sur un problème que nous ne parvenons pas à résoudre.

Souvenons-nous de cette joie que l'on éprouve quand on réussit à coder une feature qui fonctionne sans accroc ou encore qu'on parvienne à bout de ce projet qui nous faisait peur, car semblant complexe.

Cet enthousiasme lié au travail accompli témoigne du

plaisir que l'on a pu éprouver en fin de compte malgré la frustration qu'on a pu ressentir en chemin. À cela s'ajoute le plus souvent la fierté d'avoir accompli ce qu'on pensait impossible.

Ces moments de joie, de plaisir, nous les avons tous ressentis quelque soit l'intensité et indépendamment de si on est passionné ou non. Ceux-là sont universels et accessibles à tous. Ils ne dépendent pas de quelque chose de suggestif. Ils échappent à la tyrannie discriminante de la passion. Nul besoin d'être passionnée pour prendre plaisir à coder.

Le code est un terrain de jeu pour tout développeur. Chaque jour, on s'amuse tels de grands enfants à résoudre des problèmes, à concevoir des solutions adaptées et à en savourer le résultat obtenu.

Conclusion

La passion est un état affectif. C'est un paramètre suggestif. Il n'existe malheureusement aucun manuel pour développer une passion sinon cela se saurait ! La passion dépend de chacun et n'est pas forcément quelque chose qui se transmet, car être une personne passionnée n'est pas gage de pédagogie.

Personnellement, je me considère comme une développeuse passionnée. Le code n'est pas ma seule passion. Toute ma vie ne tourne pas autour du code. Je m'autorise à remettre à plus tard ce *side project* dans lequel je me suis lancée. Cette passion grandit au fur et à mesure des difficultés que je rencontre, des victoires que je célèbre et de la joie que j'ai à intervenir pour parler de ce métier. Aristote disait « Le plaisir dans le métier met la perfection dans le travail ». Alors si en lieu et place de courir après la passion, on savourait le plaisir que cela peut être de participer à la naissance d'un projet, de faire partie d'un corps de métier dans lequel la solidarité et l'entraide sont des valeurs fondamentales. Constat que l'on fait au regard ne serait-ce que du nombre de projets open source. Ne soyez pas passionnés, amusez-vous d'abord !

[1] Définition du Larousse :

<https://www.larousse.fr/dictionnaires/francais/passion/58522>

[2] « L'homme passionné par son travail n'a pas le sentiment de travailler » - L'express Philippe Laurent

[3] « Bonheur et engagement : le rôle de la passion dans le fonctionnement optimal en société » - Léa Bragoli-Barzan et Robert J. Vallerand – Revue québécoise de psychologie

[4] « Être passionné par votre travail peut se retourner contre vous » - Slate Hakim Mokadem

[5] « Understanding contemporary forms of exploitation : Attributions of passion serve to legitimize the poor treatment of workers » - Journal of Personality and Social Psychology

[6] « Faut-il être passionné pour être développeur ? » - Medhi Zed – Blog Je suis un dev

[7] Définition du Larousse -

<https://www.larousse.fr/dictionnaires/francais/plaisir/61343>

Le métier de développeur.se est sursollicité

Parce que le métier de développeur.se est sursollicité, l'accompagnement de ce métier est paradoxalement négligé. Négligé dans le sens où les organisations se restreignent parfois à deux choix : manager ou expert ? Et si le parcours carrière était autre chose qu'une suite logique d'échelons à grimper pour être dans la course à la performance ? Et si c'était possible d'avoir un choix non binaire ? Voici un retour d'expérience concret sur un parcours carrière pas parfait, mais qui a le mérite d'ouvrir le champ des possibles dans la tech.

Plutôt expert technique ou engineering manager ?

RH depuis plus de 10 ans, et coach d'équipe depuis 5 ans, j'observe régulièrement (pour ne pas dire majoritairement) un manque de vision des entreprises pour accompagner les carrières des métiers techniques. Ah bon ? Me direz-vous. Pourtant nous avons un parcours carrière et une revue annuelle avec des objectifs. Effectivement, vous avez défini des objectifs certes, mais souvent ils sont liés au produit, à l'équipe ou au rôle. Ces objectifs ne parlent pas de posture et d'évolution. D'ailleurs, on retrouve souvent deux possibilités d'évolution : expert(e) ou manager avec des niveaux pour chacune des catégories.

Le choix est facile

C'est pourtant simple, il suffit de choisir entre l'expertise ou le management. Et nous savons très bien, de toi et moi, que si tu empruntes le chemin "manager" ce sera plus "facile" pour toi d'être valorisé.e, reconnu.e ou peut être promu.e ? J'ai vu à plusieurs reprises des sociétés qui laissaient partir de très bons architectes ou tech leads parce que la seule manière de les faire évoluer, en termes d'échelon et/ou de rémunération, était de passer par la case manager. Et c'est aussi valable pour le recrutement. Encore à l'instant, je viens d'entendre une entreprise me dire qu'elle écarte les tech leads qui ne "managent pas hiérarchiquement". Et le mentoring ? "Non plus". Ah !



Pauline GARRIC

Après plus de dix ans dans les Ressources Humaines, dont sept années très structurantes dans un groupe de conseil, je deviens RH Advisor (stratégie et marque employeur) et coach au sein de BENEXT. Mordue du terrain, c'est par la pratique que j'apprends ! Depuis 2018, je suis certifiée executive coach HEC. C'est pour moi la suite logique de l'accompagnement individuel que j'ai pu faire en tant que RH. Ma vision : développer et accompagner les cadres dirigeants et sponsors qui font partie intégrante de la transformation des organisations.

J'ai une toute autre vision de l'accompagnement d'équipe. J'encadre aujourd'hui six personnes et mon positionnement est celui d'une experte. Je suis à l'écoute, je conseille, je suis à disposition pour proposer de la veille ou des idées et pour trancher quand il le faut, à leur demande autant que nécessaire. Et je n'ai pas l'impression de manquer de quelque chose ou de ne pas grandir sur mon poste. Au contraire, quelle satisfaction de pouvoir continuer à monter en compétence sur d'autres sujets transverses ou en lien avec mon expertise.

J'observe que les métiers issus de la technique se retrouvent malheureusement souvent confrontés au dilemme suivant :

- devenir manager et mettre moins les mains dans le cambouis pour espérer une augmentation
- rester sur leur expertise et atteindre rapidement un plafond de verre en termes de rémunération et de responsabilités

Pourquoi la vision de l'accompagnement d'équipe se restreint-elle à un management hiérarchique ? Pourquoi l'expertise technique n'est-elle pas autant valorisée que des compétences en management ?

Ce qui est terrible c'est qu'en étant contraintes, les entreprises se retrouvent souvent avec des personnes qui ne managent pas par envie. Et c'est courir à la catastrophe, car au final personne n'en tire vraiment parti. Ces collaborateurs se retrouvent à effectuer un management par défaut. Et la plupart du temps sans accompagnement, sans formation. Alors qu'en tant qu'expert reconnu par leur hiérarchie, ils pourraient apporter énormément à l'équipe.

Hors des cases habituelles

Vous l'avez compris, je n'aime pas les cases. D'ailleurs j'ai toujours choisi ou créé des postes hybrides. Parce que j'aime apprendre et je ne segmente pas les mondes. Je préfère avoir une vision holistique. Un monde où la posture et les compétences sont complémentaires.

J'ai donc eu à cœur de confectionner un parcours carrière pour l'ensemble des collaborateurs de benext en lien avec la technique, je l'ai construit avec l'aide de développeurs, orienté sur l'apprentissage des deux axes (expertise et management) et sur l'auto-évaluation. Ce parcours de carrière va au-delà d'un choix binaire entre manager ou expert.e et se centre non pas sur un rôle, mais sur des compétences.

Quel est le but de cette grille de lecture ? Il est multiple. Avant tout, c'est un guide pour se challenger, se fixer des objectifs et s'assurer de développer son potentiel quel que soit le niveau d'expérience actuel.

Mais aussi :

- avoir des perspectives d'évolution concrètes dans l'organisation et la liberté de dessiner son parcours
- s'auto-évaluer et se confronter au regard de ses pairs avec un référentiel commun

- prendre conscience de notre marge de progression sur les différents sujets et se challenger via des attentes concrètes et, dès que c'est possible, quantifiables
- avoir des pistes, savoir où explorer et creuser pour approfondir ses connaissances

Le modèle SHU HA RI

Avec Jordan Chapuy et d'autres issus des métiers techniques, nous nous sommes évertués à le rendre simple, encadrant et souple. Il nourrit des discussions personnelles ou en groupe, avec des pairs. Ne voulant pas de "niveau" qui impacterait l'idée initiale de grandir pour soi et non de se comparer aux autres, nous nous sommes appuyés sur un modèle existant dans l'entreprise : le SHU HA RI.

Shuhari est un concept issu des arts martiaux japonais qui décrit les 3 étapes de l'apprentissage. Il est parfois appliqué à d'autres disciplines comme le jeu de go. ShuHaRI peut se traduire par suivre les règles, comprendre les règles et transcender les règles :

- **Shu** ("protéger", "obéir") : apprendre les fondamentaux
- **Ha** ("se détacher", "digresser") : trouver les exceptions à la sagesse traditionnelle, trouver de nouvelles approches
- **Ri** ("quitter", "se séparer") : il n'y a pas de technique ou de sagesse traditionnelle, tous les mouvements sont permis

Ce framework se découpe en plusieurs onglets thématiques à l'intérieur desquels nous trouvons le concept SHU HA RI pour chacune des sous thématiques.

- un onglet socle commun qui reprend les valeurs de l'entreprise
- un onglet socle commun technique qui reprend les essentiels des bases techniques à connaître
- des onglets WEB, iOS, Android, DATA avec des focus de plus en plus précis. **Figure 1**

Ci-dessous un exemple de l'onglet socle technique commun : **Figure 2**

Chaque collaborateur aura donc le choix d'un focus (son expertise de base) et aura libre cours de choisir une ou deux saveurs qu'il ou elle souhaite développer en plus. Par exemple, mon expertise de base est le web (focus), mais je souhaite grandir sur le craft (saveur). Encore une fois, ce n'est pas une obligation. Tout est question de timing. Par moments, on souhaite sortir de sa zone de confort, et à d'autres moments, la sécurité.

La grille SHU HA RI permet à la fois de se projeter et de construire les prochaines étapes, sans pour autant rentrer dans un mode "j'ai bien coché toutes les cases".

Il nous semblait important de prendre conscience que le potentiel de chacun peut être accompagné, mais qu'il s'agit d'une responsabilité partagée : celle de l'organisation et de la personne concernée.

Et pour ne pas se limiter, nous avons ouvert aux métiers techniques la partie produit (et inversement). Ce qui fait de belles histoires, comme ce développeur qui est devenu Scrum Master.

Figure 1

	iOS	Android	Web	Data
	Socle commun			
	Socle tech			
Base	Focus iOS	Focus Android	Focus Front-End	Focus Back-End
Se développer	Focus Craft	Focus Back-End	Focus Craft	Focus Back-End
Pour aller plus loin	Focus Coaching	Focus DevOps	Focus Coaching	Focus DevOps
				Focus Data lake/warehouse
				Focus Experimentation
				Focus ML-Ops
				Focus mise en Prod

Nom	Shu	Ha	Ri
Design Pattern	Tu maîtrises les patterns essentiels : builder, factory, proxy, observer, facade, adapter, iterator, DI.	Tu connais les différentes familles. Tu connais l'ensemble des patterns classiques (même Visitor, Command, Flyweight, Chain of Responsibility ...). Tu sais quand les utiliser et ne pas les utiliser (surtout singleton :)).	Tu connais même les design patterns d'architecture et de concurrence. Tu es capable de les simplifier ou de les adapter dans un paradigme de programmation fonctionnelle. Tu accompagnes d'autres développeurs sur les Design Pattern.
Git	Tu connais les bases de git et tu l'utilises sur un projet (commit, branche, local / remote, pull, push, merge, status). Tu connais le Gitflow.	Tu sais utiliser Git en ligne de commande. Tu sais utiliser des commandes avancées comme le rebase interactif (rebase, squash). Tu connais les limites de GitFlow.	Tu as une utilisation avancée de Git. Tu sais te sortir de situations complexes avec le reflog par exemple. Submodules, subtrees, git lfs, git hooks te parlent. Tu aides d'autres personnes dans l'utilisation de Git.
Scripting	Tu maîtrises les bases de bash / shell (parcourir une arborescence, lister, déplacer, créer, supprimer, changer les droits, ...). Tu es capable de créer des scripts simples et de les exécuter.	Tu as une grande maîtrise de bash / shell et des commandes UNIX. Gestion des processus, grep, sed, awk, etc. Tu sais combiner des commandes. Tu es capable de créer des scripts shell plus complexe, avec arguments, conditions, fonctions, ... Tu sais utiliser vi / vim.	Tu maîtrises d'autres langages de scripting comme Python pour créer des scripts avancés. Tu as scripté ou automatisé plusieurs de tes tâches. Tu accompagnes d'autres personnes sur le scripting et l'automatisation.
Polyglote	Tu utilises régulièrement un langage que tu connais bien. Et tu utilises parfois quelques autres langages.	Tu maîtrises plusieurs langages dans des paradigmes différents.	Tu comprends les différents paradigmes, leurs concepts, leurs forces et faiblesses. Tu peux les expliquer à d'autres développeurs. Tu es capable d'appliquer ces concepts et t'adaptes sur le terrain en fonction de tes connaissances.
Collaboration	Tu fais des revues de code. Tu sais demander de l'aide. Tu échanges avec toute l'équipe.	Tu as régulièrement du feedback, et tu en donnes. Tu fais du pair-programming ou mob-programming.	Tu sensibilises sur l'importance de travailler ensemble, de communiquer. Tu mets en place des moments de collaboration dans ton équipe (pair-programming par exemple) ou des ateliers.
Sensibilité au craft	Tu te sensibilises au software craftsmanship. Tu as lu un livre sur le craft. Tu assistes à des meetups ou des conférences ou des enregistrements de talk. Tu as conscience de l'importance de créer du code de qualité, d'améliorer, d'itérer, de tester, de ne pas être un simple exécutant, etc.	Tu appliques au quotidien des pratiques d'artisanat du code. Le code que tu écris est testé. Tu challenges les idées, tu proposes, tu discutes avec les pairs, tu discutes avec le métier.	Voir le focus Craft.

Figure 2

Regards croisés sur un même template

Ouverture et souplesse du cadre, voilà ce qui me semble important dans l'accompagnement d'une carrière. Et cette grille, comment l'utilise-t-on ? Nous avons mis en place plusieurs moments autour de cette grille. En premier lieu, nous faisons en sorte qu'un nouveau collaborateur l'ait le plus rapidement possible pour prendre le temps de la lire et de s'auto-évaluer. Les mentors techniques proposent également d'en parler en session collective où chacun peut poser son post-it par ligne et en discuter avec ses pairs.

Les regards croisés sont importants et permettent à chacun d'ajuster ou non son auto-évaluation. Le collaborateur a aussi la possibilité de faire des one-to-one avec son ou sa mentor pour affiner sa réflexion.

Un troisième regard est proposé : celui de la "boussole". Chez benext, les RH ont un rôle de suivi important. Elles rencontrent les benexters tous les 4 mois pour les accompagner dans leur carrière et les orienter vers les bonnes personnes lorsque nécessaire. Elles ont toutes un vernis tech et le fait de ne pas être technique permet justement de poser des questions plus ouvertes et oblige à une certaine réflexion. A la fin de ces moments, appelés

"escalas", entre 3 et 4 objectifs sont pris pour accompagner ce qui s'est dit et faire le point quatre mois plus tard.

La vertu de ces différents moments est de pouvoir être accompagné à 360° tout en ayant son propre regard et la liberté de choisir au rythme qui convient.

Conclusion

Si je vous ai fait part en détail de cet exemple d'accompagnement de carrière, c'est plus en termes de philosophie. Car comme tout modèle, il est faux. Ce qui compte avant toute chose, c'est de trouver un juste équilibre entre l'envie et l'utilité, d'ouvrir un peu les chakras pour s'apercevoir que le management n'est absolument pas une fin en soi. L'expertise est quelque chose qui se travaille dans la durée, qui se forge à force de voir des problématiques différentes et pourquoi pas des contextes différents. Et cela demande autant d'investissement.

Et il y a aussi plein d'autres possibilités ; soyez créatif.ve dans votre carrière pour continuer d'apprendre et trouver du plaisir dans ce que vous faites. On passe tellement de temps au travail, ce serait dommage de passer à côté de sa passion et de ce pour quoi l'on est doué.e à cause d'une histoire de management !

1 an de Programmez!

ABONNEMENT PDF : 39 €

Abonnez-vous directement sur
www.programmez.com





Sara Dufour

Coach Agile et Scrum Master

Consultante depuis 3 ans, elle a démarré sa carrière dans la Communication et le Marketing dans des grands groupes. Elle accompagne aujourd'hui le changement dans les organisations pour améliorer l'engagement et la création de valeur, la croissance collective et individuelle, l'épanouissement au travail. Issue d'un milieu multiculturel, elle apprécie particulièrement la CNV, les thématiques liées à la diversité et l'inclusion, l'art et la gastronomie.

Anne Gabrillagues

23 ans d'XP professionnelle
Coach Agile (consultante) depuis 10 ans
Études en Éco-Gestion, Maths et Statistiques, Ingénierie logicielle
1er job : Développeuse

Aurélié Robert

20 ans d'XP professionnelle
Coach Agile, facilitatrice UX (consultante) depuis 6 ans
Études en Biologie
1er job : Testeuse

Julie Quillé

13 ans d'XP professionnelle
Coach Agile et développeuse de conscience (consultante) depuis 4 ans
Études en Langues
1er job : Chef de projet événementiel

Tiphanie Vinet

20 ans d'XP professionnelle
Coach Produit (consultante) depuis 3 ans
Études de Photographie argentique
1er job : Retoucheuse photo, technicienne numérique

Nathalie Du Four

22 ans d'XP professionnelle
Coach Agile (en interne) depuis 1 an
Études en Management
1er job : Recruteuse IT

Coaching Agile, regards croisés

Elles sont peu nombreuses sur le marché. Assez rares en freelance ou en interne. Les coachs agiles que vous croisez sont le plus souvent des hommes. Pour ce numéro spécial, j'ai voulu donner la parole à des consœurs dans l'espoir de vous faire (re)découvrir le rôle, à travers cinq visions et personnalités. Dans l'envie de donner de la visibilité et de montrer la légitimité des femmes dans le métier. Dans le but aussi de vous inviter, mesdames, à considérer cette voie. Existe-t-il un coaching agile "au féminin" ? A vous de juger à travers cette synthèse d'interviews.

Coach Agile, c'est une fonction parfois un peu floue.

On les voit organiser des quarterly plannings ou autres points de synchronisation multi-équipes, faciliter des rituels, ateliers ou sessions de mob-programming, animer des formations, bavarder (à l'époque) à la machine à café. Derrière ces actions visibles se cachent beaucoup plus de choses. Lorsque je leur demande comment elles décrivent leur métier, toutes ou presque concordent sur le point résumé par Julie : "J'accompagne les personnes et les équipes pour être autonomes". "On contribue à la transformation d'une organisation, au changement", rappelle Nathalie, qui évoque aussi la notion de performance — "en aidant les personnes à découvrir et s'approprier des façons de travailler plus efficaces", précise Anne. Aurélié explique aussi cette aide à la prise de recul : "Je m'attache à créer des espaces, une poche de temps, un lieu physique, une liberté mentale chez les personnes." Tiphanie "accompagne les équipes ou les personnes à définir le sens de ce qu'elles sont en train de faire, la vision, la manière dont on va évaluer les succès. Je les aide aussi à se mettre en ordre de marche."

Pourquoi les Coachs sont-ils utiles — pour ne pas dire indispensables dans les organisations agiles ? Ici, les réponses diffèrent : **chacune identifie un défi majeur différent au sein des organisations.**

Aurélié : "Savoir comment continuer à exister. Trouver cet équilibre entre économie et RSE, puisque les entreprises ont compris que le capital humain en était vraiment un."

Tiphanie : "Se donner vraiment les moyens de changer. Accepter que ça prend du temps, s'investir et être patient(e). Parce que ce n'est pas un sprint, c'est un marathon."

Anne : "Réussir à aller au-delà des déceptions dues aux pseudo transformations Agiles qui n'ont pas porté leurs fruits. En tant qu'équipe ou middle management blasé — souvent à juste titre — pouvoir dépasser cette appréhension négative pour trouver l'élan de (bien) changer."

Julie : "C'est d'arriver à travailler ensemble réellement. Accepter qu'il n'y a pas qu'une seule bonne solution. Accueillir un feedback 'négatif'."

Nathalie : "Pour moi c'est gérer le 'new normal'. Comment réorganiser l'hybride, après plus d'un an en télétravail ?", mais aussi "aider les équipes à comprendre la valeur de travailler en mode agile que ce soit dans le 'doing agile' ou le 'being agile'".

La matière de la ou du Coach Agile étant très "molle", sa valeur n'est pas facilement mesurable. Ses contributions peuvent sembler floues.

Pour se guider, s'assurer que les actions ont un impact — et positif — ou réajuster, les Coachs interrogées se basent toutes sur deux types de boucles de rétroaction (toujours personnalisées selon contexte et le mandat). D'une part, le déclaratif, en interrogeant régulièrement leurs clients ; d'autre part l'observation des comportements, actions, changements chez les personnes et dans l'environnement sur des critères précis. Aurélié, Anne et Tiphanie tentent au maximum de quantifier ce qui peut l'être, en fixant en amont des indicateurs extrêmement factuels (par exemple, cycle time et lead time sont les favoris d'Anne), visualisés sous forme de grilles ou de radars. Une mission démarre donc avec un cadre, posé en amont, co-construit avec le ou la sponsor.

En quoi les pratiques de ces Coachs Agiles ont-elles évolué ?

Avec le temps, place à davantage d'improvisation, de souplesse et d'adaptabilité, de liberté et, parfois, des approches plus directives. Avant, Tiphanie préparait tout. "C'était millimétré, très cadencé, et maintenant je suis de plus en plus à l'aise avec les outils et techniques, j'ai des bases solides donc j'ai besoin de moins de préparation. Je peux improviser en fonction des besoins du client le jour J." Aurélié admet en riant : "Au début on fait avec ce qu'on a, on suit le guide ! Puis au fur et à mesure on s'aperçoit des vraies contraintes des entreprises. Donc maintenant, c'est ce que je regarde en premier, je vois ce qui peut bouger à court, moyen, long terme. J'ai accepté que l'on ne peut pas tout changer du jour au lendemain, j'aide le client à prioriser les changements."

L'approche favorite d'Anne, celle du "petit pas par petit pas" ne peut pas toujours être mise en place par manque de temps ou de budget. "Je me retrouve parfois à devoir adopter des postures plus directives pour assainir une situation, jusqu'à ce que la pression baisse sur l'équipe pour revenir sur un accompagnement plus classique. C'est nouveau pour moi, ce mode directif n'est pas naturel pour moi, ce n'est pas mon approche favorite".

Au début, Julie n'osait pas du tout sortir du cadre : "Je me disais : 'Si on ne m'a pas demandé, ce n'est pas mon rôle.' Mais en fait, j'ai plus de liberté que ce que je pense. Je prends de plus en plus d'initiatives par rapport à ce qui était prévu initialement."

Sans surprise, lorsque je les interroge sur leurs plus grandes forces, la réponse commence par une hésitation, un rire ou un regard gênés. Après l'humilité, puis l'instinct et la capacité à créer rapidement un lien de confiance, à identifier des petits pas —caractéristiques communes aux cinq—, lorsque l'on creuse un peu, ces professionnelles arrivent à dégager leurs qualités utiles au métier.

Pour Nathalie, souplesse, vision transversale, facilité à identifier et capitaliser sur les forces des autres. Les talents de Julie ? "Percevoir des choses très subtiles qui se passent entre les gens, dans un groupe. Entrer en empathie, même si je n'adhère pas au contenu."

Grande capacité d'appréhension de systèmes complexes pour Anne et Tiphanie. La première cite également la curiosité et la persévérance, le goût du challenge. A chacune son style, sa saveur. Alors qu'Aurélie est très "factuelle, franche et cash", qu'on valorise sa capacité à "réenchanter le quotidien et à redonner l'élan" grâce à son dynamisme, Tiphanie mentionne sa posture basse. "On me reproche parfois de ne pas être assez confrontante mais je sais que je ne sais pas. J'écoute. Je pense que les gens attendent d'être aidés, pas jugés."

"Quelle est ta plus grande réussite ?"

Julie, experte en CNV* : "Une fois j'ai demandé à un client ce que mon accompagnement lui avait apporté. 'Sincèrement dans le boulot je sais pas, mais je crie moins sur mes enfants.'"

Anne : "Dans un contexte très complexe et tendu, commencer à travailler en Agile sous le radar, avec la complicité de l'équipe. Ça a eu des effets bénéfiques (aussi bien sur la qualité que sur l'efficacité et la réactivité de l'équipe), le client s'en est aperçu. Il nous a confié un projet supplémentaire pour tester. Ce deuxième succès a été le début d'une nouvelle vie pour cette équipe, l'Agilité s'est propagée côté métier et alors que la mission devait être brève, on a fini par y rester plusieurs années."

Tiphanie : "Réussir à davantage écouter mes intuitions."

Aurélie : "Mettre à contribution les 350 collaborateurs de notre entreprise pour co-construire un nouveau business plan il y a 3 ans, avec notamment des world cafés géants avec une centaine de personnes."

Nathalie : "Avoir réussi dans plusieurs métiers différents, être passée de l'un à l'autre, et voir que certaines choses mises en place marchent toujours ou ont évolué dans le bon sens."

* Communication Non-Violente

Coach et femme, ça t'évoque quoi ?

"Rien, ce n'est pas le sexe qui différencie les individus. Il y a des milieux très masculins et virils qui sont aussi des challenges pour certains de mes homologues masculins. Ce n'est pas parce que je suis une femme que je n'y suis pas bienvenue et que je n'y ai pas ma place —pour le coup ma différence de genre devient une plus-value pour voir les choses différemment."

Nathalie semble gênée. "Je n'aime pas trop la question parce que ça peut stigmatiser", avant de partager qu'elle estime que c'est une force. "J'ignore si c'est par ma personnalité ou le fait d'être une femme —probablement les deux— mais j'ai l'impression que j'arrive à faire passer beaucoup plus de choses dans des groupes de décideurs (majoritairement masculins), que si j'étais un homme. Je pense que les hommes voient moins les femmes comme des adversaires, ils se sentent moins menacés donc sont plus ouverts, et je peux dire franchement les choses."

Pour Tiphanie, ça n'a jamais été un problème. "J'en parle avec des copines, je vois qu'elles ont dû se battre. 'Moi, j'ai eu de la chance, j'ai grandi dans une entreprise féminine et féministe. Ensuite, dans mon ancienne ESN, c'était différent (90% d'hommes). Les mecs ont la tchatche. Moi, je fais. Je délivre beaucoup de contenu ; en général ça force leur respect.'"

"Je vais être honnête : dans le passé, j'ai été confrontée à des situations fortement entachées de sexisme (grossier ou subtil)", reconnaît Anne. "Étonnamment, maintenant que je suis coach, j'ai l'impression d'être moins exposée à ce genre de problèmes. Dans notre métier, les coachs femmes sont généralement plus axées sur le coaching professionnel, le développement personnel... Mes interlocuteurs sont surpris quand je parle d'architecture, d'ingénierie logicielle, etc. Je dois parfois un peu plus m'imposer pour démontrer que j'ai des compétences techniques, je fais face à des réactions ou remarques auxquelles mes collègues masculins ne sont pas forcément confrontés."

"Femme et coach... ou femme tout court ?", questionne Julie avec un sourire complice. "J'ai l'impression qu'en entreprise, on mesure souvent les individus à partir de leur capacité à prendre l'espace, à s'affirmer, voire à s'imposer. Ces jeux de domination peuvent être un challenge pour grimper les échelons, ça peut être difficile de concilier ce qu'on est ou veut être et ce qui est attendu de nous. Et pourtant, l'Agilité offre beaucoup plus de place à la diversité..."

Que feront-elles dans 5, 10 ans ?

Aucune ne le sait, elles n'ont pas de plan tout tracé. Même si elles adorent leur métier, elles sont toutes cinq ouvertes au changement, aux opportunités. Elles sauront écouter leur instinct, se laisser porter. Et peut-être seront-elles rejointes par une génération d'anciennes développeuses prêtes à féminiser le métier, pour ainsi créer de nouveaux espaces d'innovation et d'inclusion ?

CINQ LIVRES POUR DÉMARRER

Coaching Agile Teams
de Lyssa Adkins

Comment réussir à travailler avec presque tout le monde
de Lucy Gill

Scrum depuis les tranchées
d'Henrik Kniberg

Les mots sont des fenêtres, ou bien ils sont des murs
de Marshall Rosenberg

Solution Focus
de Géry Derbier



Samira Aziki

*Comptable en entreprise
durant une quinzaine
d'années, et
actuellement
développeuse front-end
et chef de projet digital*

REX : Bootcamp de 3 mois et l'expérience que j'en retire

Il y a un an, j'ai pris un virage à 360° pour intégrer le monde de la tech et enfin faire un métier où je pourrais m'épanouir. J'ai donc suivi un bootcamp de 3 mois, intense et très formateur. Aujourd'hui je développe des sites web et termine une formation de Chef de projet Digital.

A l'automne 2020, j'ai fait un bootcamp qui formait au métier de développeur fullstack car j'avais pour projet de faire une reconversion professionnelle et me réorienter vers le développement web. Je vais vous raconter cette aventure.

Mon parcours

Jusqu'à peu de temps, j'étais comptable au sein d'un service financier dans un grand groupe comptable. Durant 15 ans, j'ai tenu différents postes de comptable mais je ne me suis jamais sentie épanouie dans mon travail. Je m'ennuyais vite et c'est une des raisons pour laquelle j'ai fait de l'intérim durant 8 ans, ce qui me permettait de changer d'entreprise, d'environnement, de collègues, d'activité mais malgré cela je ne me sentais jamais à ma place et j'ai compris que le poste idéal n'existait pas. Passé la trentaine, j'avais besoin de stabilité pour raisons personnelles et j'ai donc accepté mon premier CDI et j'y suis resté presque 4 ans, ce qui me semblait une éternité. J'ai quitté ce poste pour un autre CDI dans un grand groupe où je m'étais promis de rester très longtemps car le poste offrait pleins d'avantages mais là encore, au bout de 4 ans, j'avais la bougeotte et je m'ennuyais à mourir. J'ai donc voulu écrire un blog sur la cuisine mais le web m'était inconnu et c'est en faisant des recherches que j'ai découvert un monde magique qui me parlait de wordpress, de html, css et que je trouvais passionnant ! De fil en aiguille, j'ai découvert le métier de développeur qui s'apparente pour moi à des magiciens : écrire des lignes de codes et voir apparaître une fonctionnalité, une couleur, des images, c'est juste magique ! J'avais envie d'en savoir plus sur ce métier, et plus j'en apprenais, plus mon job de comptable me semblait morne et terne. Le confinement a sonné le glas de ce poste car je n'ai jamais voulu y retourner après les vacances d'été et j'ai demandé une rupture conventionnelle. Et me voilà en septembre, sans travail, mais avec la ferme intention d'apprendre le développement web. J'ai donc commencé à chercher comment me former rapidement car ayant atteint la quarantaine, je ne pouvais pas me permettre de faire de longues études et j'avais besoin de me reconverter rapidement. Je me suis donc inscrite à un bootcamp en

ligne un peu par hasard et qui promettait de devenir développeur full-stack à l'issue des 3 mois de formation

Le Bootcamp

Je n'avais aucune idée des différents langages qui pouvaient exister et je ne connaissais rien au code. J'avais juste appris seule ce qu'était le HTML, le CSS et à faire une page statique. Et pour moi, ça représentait déjà une prouesse.

J'appréhendais un peu la formation car ne venant pas du tout du monde informatique, je ne savais pas trop à quoi m'attendre. Surtout qu'après avoir été comptable pendant 15 ans, j'avais été quelque peu formatée par le monde de la comptabilité d'entreprise, ma façon de penser était complètement différente, et je venais d'un métier où tout était normé et les processus d'entreprise dictaient notre façon de travailler. Je me suis retrouvée propulsée dans un domaine qui laissait place à l'imagination, où il n'existait pas qu'un seul chemin pour aboutir à une solution et j'avoue que cela m'a un peu perturbé au début. Comment réussir à m'adapter à un nouvel environnement, une nouvelle façon de travailler, un nouveau domaine à mon âge ?

Je me disais que j'allais me retrouver avec des jeunots qui comprendraient tout très rapidement et que je serais la "vieille" dans cette formation, les Bootcamps ayant la réputation d'être très hard à suivre et qu'il fallait bien s'accrocher. Mais malgré mes doutes et mes appréhensions, j'étais très motivée. C'était un vrai challenge à relever mais j'aime les défis et je me disais que je n'avais rien à perdre. Et j'ai bien fait de m'écouter car j'ai été agréablement surprise de voir qu'il y avait tous les profils et tous les niveaux. Beaucoup d'apprenants étaient en reconversion et je peux même dire que c'était les plus motivés.

La formation se déroulait en peer-to-peer, c'est-à-dire qu'il n'y avait pas de professeurs à proprement parler mais que l'apprentissage se faisait grâce aux connaissances des autres. Nous avions chaque matin une ressource à travailler et si nous avions des difficultés pour comprendre le concept présenté, nous devions faire des recherches par nous-même pour trouver les informations ou demander aux autres apprenants de nous aider. C'était très collaboratif et vivant, cette technique d'apprentissage favorisant vraiment l'échange et le travail d'équipe.

J'avais dans l'idée qu'en faisant ce Bootcamp, j'allais me retrouver seul devant mon pc mais ce fut tout le contraire : je communiquais toute la journée avec les autres pour comprendre les ressources et parfois j'étais fière d'expliquer certains concepts que j'avais compris.

La plupart du temps, je travaillais en binôme avec une autre fille, Marine, et on s'est motivées mutuellement tout le long de cette formation pour ne rien lâcher et ce fut très riche, tant techniquement qu'humainement.

Deux fois par semaine, le mardi et jeudi, nous avions des exercices à réaliser sur les ressources de la veille et le mercredi et vendredi, deux correcteurs nous étaient attribués auxquels on présentait notre travail.

De notre côté, nous devions également corriger deux autres personnes, ce qui nous permettait de faire connaissance avec d'autres apprenants venant d'horizons totalement différents, d'ouvrir notre réflexion et de voir qu'il n'y avait pas qu'une seule solution à l'exercice, chacun appréhendant le problème de façon différente.

La Formation

Le programme était très intensif et se déroulait sur 12 semaines. Il nécessitait d'être disponible du matin au soir, du lundi au vendredi. On accédait à la formation via un espace personnel sur la plateforme du Bootcamp et les échanges avec les autres élèves se faisaient sur un Discord dédié sur lequel des groupes de travail s'étaient librement composés en fonction des affinités et on pouvait intégrer celui de notre choix sans problème. Il y avait beaucoup de solidarité entre les apprenants et cela m'a agréablement surpris. J'étais restée sur cette ambiance de service comptable où on se tirait dans les pattes et où l'erreur n'était pas permise. Avec ce bootcamp, j'avais le droit de ne pas savoir, de ne pas comprendre, de me tromper et cela faisait du bien et me motivait à en apprendre plus. J'ai adoré cette bienveillance entre apprenants !

La formation proposait le javascript pour le Front, Ruby On Rails pour le Back et SQL pour les bases de données. La première partie était composée d'une introduction au code qui se déroulait sur trois semaines. Durant ces trois semaines, nous avons découvert tout l'environnement du développement web ; ce qu'était Git et Github, à utiliser le terminal, ce qu'est l'atomic design et comment faire sa veille technologique, point important quand on veut approfondir ses connaissances. Nous avons également appris ce qu'était une boucle, un array et tous les termes indispensables à l'apprentissage d'un langage de développement. Cette période d'initiation se terminait par la découverte de Ruby où nous avions à faire beaucoup d'exercices pour nous habituer au langage et à mettre en pratique ce qu'on a appris durant ces trois semaines.

L'essentiel de la formation se déroulait sur les neuf semaines suivantes et était scindé en trois parties d'une durée de trois semaines chacune :

- La première partie était réservée au BackEnd où nous avons appris à programmer avec Ruby On Rails. Nous avons pu apprendre ce qu'est la Programmation Orientée Objet, le concept MVC ainsi que la création de bases de données relationnelles et la communication avec la BDD grâce au langage SQL. Nous avons également appris à utiliser Heroku et les différents concepts de Rails. Ce qui nous a permis de mettre en ligne notre premier site web.

Et tout cela en trois semaines. Cela vous laisse imaginer le rythme effréné et l'intensité de la formation.

- La seconde partie était plus tournée vers le FrontEnd avec une révision plus approfondie du HTML et CSS vu en début de formation. Nous avons également appris à intégrer un thème et à utiliser Ajax.
- La troisième partie consistait à travailler en équipe de 4 personnes et il fallait créer une petite boutique e-commerce durant la première semaine et un projet final à réaliser et à présenter devant un jury composé de trois professionnels du développement Web à l'issue des deux semaines restantes. Cette dernière partie permettait de mettre en pratique tout ce que nous avons appris durant la formation mais également de savoir travailler en équipe et d'appréhender la gestion de projet via des outils tels que Trello.

Ce fut très intéressant car nous avons pu travailler en situation réelle, avec une deadline à respecter et nous avons pu découvrir la gestion de projet en situation réelle. De plus, le fait de savoir que nous serions évaluées par des professionnelles ajoutait un stress au projet final car c'est une chose d'être évalué par ses pairs durant la formation, s'en est une autre d'être évalué par des professionnels.

Conclusion

Je suis vraiment sortie de ma zone de confort et rien que pour ça, je ne le regrette pas. Certes, 12 semaines c'est court pour devenir développeuse web full stack et qu'il vaut mieux continuer à se former derrière mais cela reste un excellent moyen de mettre le pied à l'étrier et de découvrir d'autres métiers du web. Ce qui fut mon cas puisque faire ce bootcamp m'a ouvert d'autres voies que je n'imaginai même pas puisque aujourd'hui je développe des sites web à mon compte et que je termine une formation pour devenir cheffe de projet digital.

Je ne regrette aucunement d'avoir suivi cette formation. Elle fut riche et intense, tant par l'apprentissage du développement web qu'humainement. Ce fut une aventure humaine formidable où j'ai rencontré des gens de tous âges et venant d'horizons totalement différents. J'ai pu voir également que le monde de la tech est de plus en plus ouvert aux filles et qu'elles ont pleinement leur place dans ce monde dit "masculin".

Donc les filles, surtout n'hésitez pas, c'est que du positif et j'espère que mon témoignage pourra faire tomber ses barrières de l'âge, de genre ou de niveau.

PROCHAINS NUMÉROS

Programmez! Spécial
100 % jeux vidéo
11 février 2022

Programmez! n°251
4 mars 2022
JavaScript,
Quantique, Ruby

Les enjeux de la féminisation des métiers du numérique

Le 25 novembre dernier, Epitech, référence de l'enseignement supérieur en expertise informatique, digital et innovation, présentait à l'Assemblée nationale une étude réalisée en collaboration avec l'institut de sondage Ipsos sur la féminisation des métiers du numérique. Le constat est implacable : 9 lycéens sur 10 pensent que les femmes diplômées d'écoles d'informatique sont désavantagées par rapport aux hommes.



Emmanuel Carli
Directeur général
d'Epitech

Pour Programmez!, Emmanuel Carli, directeur général d'Epitech, Cécile Duvigneau, directrice pédagogique d'Epitech Technology à Bordeaux et Chloé Devoyod, directrice pédagogique adjointe d'Epitech Technology à Lille, sont revenus sur cette étude afin d'apporter de la perspective sur un sujet à la fois éducatif, mais aussi sociétal, et surtout réfléchir aux solutions pour le futur.

Quelle est votre perception aujourd'hui de la place des femmes dans les métiers de la tech ?

Emmanuel Carli : Elle est sans conteste trop faible. La proportion des femmes dans le numérique est très en deçà de leur place dans d'autres métiers. Ce n'est même pas un élément de discussion aujourd'hui, le constat est facile à faire.

Cécile Duvigneau : Les femmes sont clairement en sous-représentation dans les métiers de la tech alors qu'elles sont tout aussi compétentes que les hommes. Le problème est que cette vision n'est pas partagée par tous. En effet, beaucoup d'hommes ont tendance à penser que le travail des collègues femmes n'est pas assez abouti : des réflexions sur le travail rendu, une pseudo-compétition sur « qui fait le meilleur code ». On peut parfois retrouver ce genre de comportement chez les étudiants, ce que nous nous attachons à faire évoluer. Les femmes, quant à elles, n'ont souvent pas confiance en leur travail et ne vont donc pas forcément le mettre



Cécile Duvigneau
Directrice pédagogique
adjointe d'Epitech
Technology à Bordeaux

en avant, au contraire de leurs collègues masculins.

Chloé Devoyod : Néanmoins, on peut observer un changement progressif des mentalités et, bien que peu nombreuses, nous sommes de plus en plus (et heureusement !) des professionnelles comme les autres.

Quelles sont, de votre point de vue, les raisons de cette faible représentation ?

Chloé D. : Pendant très longtemps et à l'échelle de la société, certains métiers étaient réservés aux hommes, d'autres aux femmes, à tel point que peu de personnes se questionnaient à ce sujet. Les métiers de la tech en ont fait partie, et il est difficile de faire évoluer les mentalités : si les tendances de fond sont visibles à l'échelle de la population, les changements doivent se faire dans chaque école, dans chaque famille, à l'échelle individuelle. C'est un cercle vicieux : vous voyez peu de filles/femmes dans une formation/un domaine, vous vous questionnez automatiquement : est-ce que c'est "normal", est-ce fait pour moi, est-ce que j'y serai à ma place...

Cécile D. : Le problème n'est pas seulement de savoir pourquoi les femmes sont sous-représentées dans la tech, il est plus large : pourquoi les femmes sont-elles si sous-représentées dans les postes à responsabilité ? S'il n'y a pas de femmes à des postes à responsabilité, c'est qu'elles ne sont pas légitimes dans ces postes, et



Chloé Devoyod
Directrice pédagogique
adjointe d'Epitech
Technology

donc par extension dans le domaine en question ?

La carrière des femmes prend très souvent un tournant avec la maternité, donc finalement assez tôt dans la carrière d'une femme. Il faut changer les mentalités en profondeur pour que la carrière des femmes puisse évoluer comme celle des hommes. La génération actuelle des 30/40 ans, femmes et hommes confondus, connaît l'importance d'un équilibre vie pro/vie perso. Cela va influencer sur la place des femmes dans les entreprises, et le monde de l'entreprise est en train d'évoluer sur ces questions.

Emmanuel C. : Aujourd'hui on parle beaucoup du Tech for good, c'est quelque chose que nous avons initié dès 2014 à Epitech en faisant des modifications dans le parcours pédagogique, pensant que ce positionnement de l'école permettrait de faire venir les femmes. L'effet a été très limité puisque nous sommes passés de 4 à 6% d'étudiantes, sur la masse globale, c'est ridicule. Au fil des années, je me suis rendu compte au travers des Journées Portes Ouvertes qu'on sous-estimait le rôle des parents, et comment ils reportent, soit des verrous qu'ils ont eux-mêmes, parce qu'ils voient un environnement très masculin, mais aussi des verrous scientifiques. Et surtout, le combat ne commence pas dans les universités, il commence bien avant, dans les familles, dans les lycées. Quand les garçons font des remarques aux filles, cela n'est pas

l'école d'informatique où ils sont qui est le biais, mais ce qu'il y a eu avant.

L'étude Ipsos initiée par Epitech révèle justement que les principaux freins sont la perception de leur niveau dans les matières scientifiques par les jeunes filles et la méconnaissance des métiers et perspectives dans ces domaines (par elles-mêmes, mais aussi leurs parents, premiers prescripteurs).

Cécile D. : Il faut sensibiliser les enfants et les parents à la multiplicité des métiers de la tech. On peut être développeur dans l'industrie, dans la santé, dans la finance... Pas seulement dans la cybersécurité pour devenir hacker ! La multiplicité des domaines d'application des métiers du numérique doit encourager les jeunes filles à s'y intéresser. Il faut aussi que les parents aient connaissance de cela pour pouvoir aider au mieux leurs filles dans leur orientation.

Chloé D. : Il est évident que la représentation compte beaucoup : quand on voit des femmes à des postes tech, des postes responsables, des postes scientifiques, on se projette beaucoup plus dans ces domaines. Ce sont les modèles que nous avons qui façonnent notre projection et notre trajectoire, et ce bien avant que nous soyons en âge de nous orienter dans nos études !

Emmanuel C. : On a l'impression qu'il faut savoir faire des maths pour faire de l'informatique, ce qui a été contredit encore récemment par des travaux du MIT. Or les filles, qui globalement réussissent mieux à l'école, se mettent plus de verrous sur leurs capacités à réussir dans des filières scientifiques et encore plus informatiques. Ce sont des choses qu'il faut casser une fois pour toutes. Et les parents ont un rôle énorme à jouer à ce niveau-là.

Quel rôle les écoles peuvent-elles jouer pour créer une dynamique plus égalitaire ?

Chloé D. : Les écoles jouent un rôle essentiel dans cette ouverture, cette dynamique, en donnant justement ces modèles dont nous avons besoin, en ouvrant les portes et les perspectives, en faisant prendre conscience que c'est un avenir aussi viable que n'importe quel autre.

Emmanuel C. : Nous sommes à un point de convergence : les jeunes qui viennent réaliser leurs études, les parents qui suivent leurs enfants, les entreprises qui viennent les recruter, etc. Ce qu'on

observe, c'est que la faible représentation des femmes est liée à une conjonction de choses :

- Un sujet d'empowerment, avec un problème d'estime de soi quant à sa capacité à « faire ».
- Le fait que les parents voient l'entrée dans ces filières comme un « problème » et non pas comme une opportunité absolument dingue.
- La relation dans les lycées où les jeunes hommes se sentent obligés de faire des remarques aux jeunes femmes.
- L'importance de faire de la réassurance, en invitant les entreprises et les recruteurs pour montrer les opportunités.

Cécile D. : Les écoles ont plusieurs leviers d'action. Le premier est d'intervenir dès le collège pour présenter les métiers du numérique avec des alumni hommes et femmes à part égale. Dans les salons réservés à l'orientation, il faut aussi qu'il y ait plus de mixité. Le deuxième levier est sur l'encadrement de l'école : il faut une vraie diversité pour que ce ne soit pas un frein pour les jeunes filles qui n'oseraient pas intégrer une école encadrée essentiellement par des hommes. Le dernier levier est de proposer aux lycéens et étudiants des conférences sur les discriminations et le sexisme.

Quelles sont les autres actions possibles pour accélérer la mixité de ces professions, plus particulièrement dans les entreprises ?

Chloé D. : Je ne suis pas très à l'aise avec la discrimination positive : elle est parfois essentielle pour initier un changement, mais attention à ne pas créer encore plus de distance et d'isolement : les femmes ont leur place dans ces professions et ces entreprises, non pas parce qu'elles sont des femmes, mais bien parce qu'elles en ont les compétences. Pour que plus de femmes aient accès à ces entreprises et y soient légitimes, elles doivent mécaniquement être plus nombreuses en école.

Cécile D. : Il faut que les postes à responsabilité soient plus représentés par des femmes - quel que soit le domaine d'activité d'ailleurs - et que les entreprises communiquent largement sur ces femmes. À l'instar des conférences sur les discriminations et le sexisme à proposer dans les écoles, il faut également aborder ces sujets en entreprise, via des conférences ou du coaching.

La mise en avant de modèles féminins est-elle cruciale pour inspirer les jeunes femmes et évangéliser autour de la mixité de ces métiers ?

Chloé D. : Cette mise en avant est essentielle, tant qu'elle ne tombe pas dans la caricature du "je fais ce métier parce que je suis une femme". Pour moi, pour que l'impact soit le plus fort, les modèles féminins doivent avant tout être des professionnelles et présenter comme telles. C'est par leur expertise et leur légitimité technique qu'elles donneront des perspectives aux jeunes filles qui se projettent dans ces métiers.

Cécile D. : Il faut inspirer les femmes en cassant le role model : un informaticien aujourd'hui, ce n'est plus un geek qui reste enfermé dans le noir pour coder. De très nombreuses femmes ont réussi dans la tech, dans l'innovation, dans les métiers du digital. Il faut les mettre en avant, elles sont des exemples pour nos jeunes filles. Lors de choix d'un métier, on se rapproche des exemples, et on s'identifie à des personnes dans le domaine. Cette identification se fait de plusieurs manières pour chacun, mais la mixité des personnages est un atout de projection crucial.

Pourquoi est-ce un enjeu pour le futur ?

Cécile D. : Mettre en avant ces modèles féminins et favoriser la mixité dans les métiers de la tech est un enjeu pour la société que nous souhaitons construire. Une société plus égalitaire, une société où les petites filles auront un avenir tout aussi ouvert que celui des petits garçons.

Chloé D. : La mixité n'est pas un enjeu pour le futur, mais pour le présent. À l'heure où l'informatique, la technologie, le numérique est partout, tout le monde en est utilisateur et il y a autant de façon de l'aborder que d'individu. C'est en challengeant nos idées avec d'autres personnes qui ont un autre point de vue, d'autres référentiels, un autre schéma de pensée, d'autres constructions mentales que ces idées peuvent s'épanouir et donner leur plein potentiel.

Emmanuel C. : Les entreprises ont besoin d'avoir une diversité de points de vue. Il n'y a rien de plus important que la contradiction : on peut dire « oui », on peut dire « non », mais on ne peut pas ne pas écouter. L'amélioration continue est forcément un processus de rencontre, d'échange, d'ouverture : face à problème complexe, il faut une équipe variée pour apporter les bonnes solutions.



**Clémence
PITEAU**

Product designeuse chez Zenika depuis 2017, référente de l'Antenne Lyonnaise de l'association des Designers Éthiques et passionnée par les sciences cognitives, l'intelligence émotionnelle et le design systémique.



Designeuse, j'ai fait de l'humanitaire chez mon client

Je suis product designeuse depuis quelques années, et ce qui m'importe dans mon travail est la satisfaction des humains lorsqu'ils utilisent l'outil que je conçois. Dans mon métier, concevoir un produit numérique se fait toujours de manière collective avec les développeurs, les utilisateurs et l'équipe que nous nommons "la team conception". Mon métier consiste à appliquer les méthodes du design thinking⁽¹⁾ pour déceler les moments de frustrations dans le quotidien des gens et de tout mettre en œuvre pour les résoudre. Un jour, je me suis retrouvée à pratiquer ces méthodes non pas pour réaliser une interface (IHM), mais pour créer des solutions humaines. Ici, nous n'allons pas parler de développement informatique, mais bien de chair et d'os.

Je ne vais sûrement pas te l'apprendre, mais fabriquer un produit qualitatif par sa durabilité, son efficacité et sa pertinence relèvent du défi. Au vu des enjeux écologiques et humains actuels, de nombreux concepts émergent comme l'écoconception, le design systémique, le design inclusif, le green-IT, l'intersectionnalité, etc. Tous ces sujets ont un point commun avec mon métier ainsi que mes valeurs : ils veulent un monde meilleur pour les humains. Comme l'UX design, ces concepts ne sont pas encore mis en pratique dans les entreprises, alors qu'ils sont essentiels dans l'élaboration de produits numériques (ex. : intranet, site web, application mobile, etc.). Parfois, il m'arrive même d'entendre que l'UX design est un luxe. Fatalement, majoritairement ce qui guide les enjeux informatiques - et sociétaux - est le temps donc l'argent.

Pourquoi ne réussirais-je pas à rendre le monde meilleur à mon échelle ? Est-ce qu'une UX designeuse, intégrée dans une équipe de 30 personnes dans une entreprise où la politique dicte les directives, ne pourrait-elle pas améliorer les choses ?

La notion des 3 strates CEU (Client - Équipe - Utilisateur)

Avant de vouloir sauver le monde, il me fallait déjà comprendre dans quel monde nous vivons. Pour cela, j'ai pris conscience de mon quotidien, le monde réel, et j'ai ouvert les yeux pour observer ce qui m'entourait. Tout commença le jour où j'ai poussé une porte, celle de l'entreprise dans laquelle j'allais travailler pour la première fois, et ce, pendant 3 ans. Dans mon sac à dos, il y avait mon nécessaire vital : des post-it, des crayons pour dessiner des wireframes, mon carnet, mes sachets de thé et mon ordinateur. Telle une élève à la rentrée des classes, j'avais la volonté et la détermination de vouloir bien faire mon travail. C'est après avoir passé dix minutes à traverser deux couloirs, quatre étages et un grand open space que j'arrive enfin dans la zone dédiée à mon équipe avec qui j'allais concevoir une application mobile. Je n'étais pas grand-chose dans toute cette foule. Pour que tu comprennes l'étendue du projet, je vais te raconter une histoire, celle de deux jeunes aventurières en expédition.

Alors que certains êtres puissants et remplis d'ambitions préparent leur nouvelle invention révolutionnaire, nous devons explorer les étendues de terres inconnues où vivent des populations fâchées. Nous devons ramener des preuves de notre étude auprès de notre équipe pour les aider à concevoir le produit imposé par nos pairs et approuvé par ce peuple étranger. Le lieu de départ n'a pas réellement d'importance pour nous. Il n'y a qu'une seule porte à nos côtés. Alors pas le choix, nous l'empruntons. Pour glorifier l'histoire, pourrions-nous même dire que nous sommes curieuses et courageuses ? Allez, disons ça. C'est alors que nous découvrons une petite pièce où un sablier est posé sur une table. Des photos et schémas recouvrent une partie des murs. On dirait que toutes ces choses sont là depuis un moment. Trois personnes sont installées dans des fauteuils, cachées derrière leurs ordinateurs avec à leurs pieds des sacs remplis de billets verts. Qui sont-ils ? Nous n'avons pas le temps de chercher à comprendre, il faut avancer. Ne t'en fais pas, nous pouvons marcher sans crainte, personne ne nous voit. En continuant, nous trouvons une porte à hublot. Je regarde à travers, mais ne distingue rien, car la vitre est floue. À nos risques et périls, nous ouvrons et arrivons dans un long couloir encombré de dossiers, de courriers et de messages codés. Et c'est au bout d'une dizaine de minutes à enjamber les affaires au sol que nous découvrons une porte sur le côté. Elle avait une encoche sur le devant, certainement pour y déposer des documents comme les boîtes aux lettres. Qui y a-t-il derrière ?

Nous ouvrons la porte et entrons. Quel endroit immense et bruyant ! Jamais nous n'aurions pu imaginer qu'une telle civilisation puisse vivre ici. Des groupes sont formés à droite et à gauche. Certaines personnes ne semblent pas se comprendre, ils ne doivent pas parler la même langue. De là où nous sommes, impossible de cerner distinctement ce qu'ils disent et quelle langue ils parlent. Qui sont-ils d'ailleurs ? Que font-ils ? Que veulent-ils ? Le lieu est si grand. Nous marchons en direction opposée. Quelqu'un nous regarde avec insistance ce qui signifie que dans ce lieu nous ne passons pas inaperçues. En observant autour, nous voyons des instruments scientifiques, des outils, de l'artillerie et tout un tas de choses de part et d'autre. Nous pourrions y rester des heures que nous n'aurions pas tout vu. Nous faisons encore quelques pas avant de nous arrêter devant une grande sculpture. Il s'agit du même sablier que nous avons vu au début de l'aventure. Lorsqu'on voit sa gigantesque taille, il nous est impossible d'oublier que le temps nous est compté. Mais que représente le temps ? Alors que nous repérons une porte cadenassée, une personne vient nous parler. Celle-ci semble inquiète et gesticule dans tous les sens. Nous comprenons qu'elle essaye de nous convaincre de rebrousser chemin. Alors, allons-nous baisser les bras ? N'avons-nous pas dit plus haut que nous étions courageuses ? Alors, brisons le cadenas et aventurons-nous encore dans l'inconnu !

(1) Qu'est-ce que le design thinking : " Le design thinking (littéralement « penser le design »), en français démarche design ou conception créative, est une méthode de gestion de l'innovation. Il se veut une synthèse entre pensée analytique et pensée intuitive. Il fait partie d'une démarche globale appelée design collaboratif. Il s'appuie en grande partie sur un processus de co-créativité impliquant des retours de l'utilisateur final. Contrairement à la pensée analytique, le design thinking est un ensemble d'espaces qui s'entrecroisent plutôt qu'un processus linéaire ayant un début et une fin. " Wikipédia

Nous y voilà. Cet endroit ressemble faiblement à ce que nous avons vu sur les photos au début de l'aventure. Ce sont les fameuses terres inconnues. Si ce que nous avons vu jusque-là était immense, alors ici représente un univers infini. C'est là que nous devons travailler dans un premier temps. Rencontrons les gens qui vivent ici, observons-les, posons-leur des questions et découvrons ce dont ils ont besoin avec notre regard empathique. Que se passera-t-il lorsqu'ils auront cet objet révolutionnaire entre les mains ?

As-tu remarqué l'allusion à ces 3 niveaux, que je nomme des "strates", dans cette histoire ? Tout d'abord, les personnes assises derrière leurs bureaux dans la première pièce sont nos **clients** ou nos commanditaires. Ils ont compris qu'ils pourraient gagner de l'argent grâce à de nouvelles fonctionnalités regroupées dans un nouveau produit ou un ancien à améliorer. La deuxième strate correspond à ce lieu immense où nous avons vu ce grand sablier. C'est à ce niveau que **l'équipe** réalise le produit à l'aide de multiples connaissances et expertises techniques, mais pas que. Enfin, les terres inconnues représentent notre dernière strate. C'est le lieu où nous retrouvons tous les **utilisateurs** qui se servent du produit que nous concevons.

Tu l'as sûrement compris, normalement mon métier est l'exploration de la dernière strate, celle des utilisateurs finaux. Pour cela, j'utilise des méthodologies d'analyses, de conception et d'évaluation que tu découvriras dans la suite de l'article. Ainsi je me questionne : pourquoi devons-nous simplement explorer cette strate lorsque l'on voit à quel point les autres niveaux sont tout aussi incompris et complexes ? Crois-tu que les objectifs et les enjeux de chaque personne présente dans chaque strate soient les mêmes ? Nous serions naïfs de le croire. Et nous le serions encore plus de croire que cela n'a pas d'impact sur le fonctionnement du projet et par conséquent sur la fabrication de ce produit.

Par exemple, l'objectif pour la strate du client est de voir son business fleurir, de pouvoir tracer les gens qui travaillent pour eux, d'accélérer la production, etc. Toutefois, individuellement une personne appartenant au niveau client peut avoir des besoins spécifiques à lui-même. Peut-être que pendant que l'un veut faire plaisir aux utilisateurs, l'autre peut être hermétique aux enjeux émotionnels et ainsi agir à l'encontre du bien-être commun pour atteindre son objectif. Il est intéressant de porter cette réflexion au sein même de l'équipe qui conçoit. Quelles sont les attentes, besoins et frustrations des développeurs ou des product owners ? Le nombre de réponses est probablement identique au nombre de personnes dans l'équipe au global. **Figure 1**

Et s'il fallait voir l'équipe comme un produit que l'on façonne ?

Puisque nous ne pouvons pas mener toutes les batailles en même temps, je vais poursuivre mon histoire en parlant de mon équipe. As-tu déjà entendu quelqu'un dire : "mon projet s'est toujours super bien déroulé" ou "Il n'y a jamais eu aucun problème dans mon équipe" ? Je prends le risque de conclure que cela n'arrive presque jamais. D'ailleurs, ce n'est peut-être

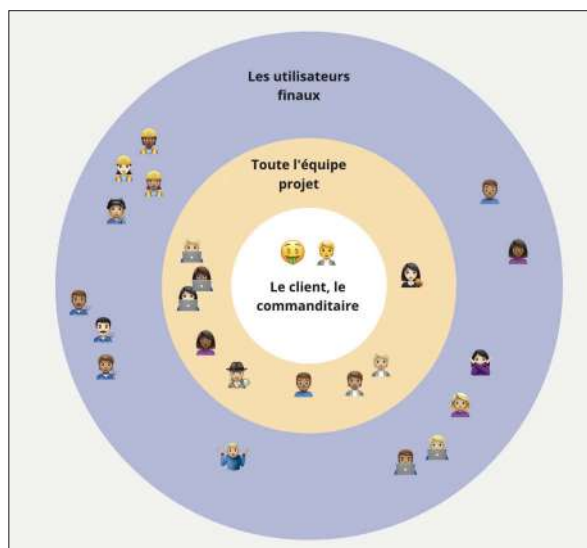


Figure 1

pas un hasard si les projets qui fonctionnent en agilité organisent toutes les deux ou trois semaines des réunions que l'on appelle des rétrospectives ? C'est bien une preuve qu'il y a souvent des choses à revoir au sein des équipes.

La phase d'analyse

Mon équipe avait déjà des difficultés lorsque je suis arrivée et mon premier réflexe, par déformation professionnelle, était d'écouter et d'observer les gens qui exprimaient leurs points de vue. Pour cela, je prenais le temps de discuter lors des pauses ou pendant le repas du midi. Il m'arrivait aussi de poser des questions et cela était souvent perçu comme de la curiosité. J'ai même dû attendre la rétrospective suivante pour entendre la voix de la totalité de l'équipe, car les tempéraments de chacun.e sont différents et il faut en tenir compte. Plus simplement, je regardais les gens agir dans l'open space. Leur comportement en disait long sur leur état d'esprit et sur leur aisance vis-à-vis des uns et des autres. C'est alors que j'ai compris que j'étais dans la phase exploratoire qui est une étape clé dans le processus de recherche utilisateur.

Durant cette étape nous récoltons des données en faisant des immersions terrain, des interviews, des audits de l'existant et des sondages auprès d'utilisateurs potentiels. Puis avec une partie des données, nous créons un schéma racontant la journée type de chaque archétype d'utilisateur. Dans mon cas, mes utilisateurs étaient des développeurs fullstack, des développeurs backend, des frontend, un lead dev, un intégrateur, deux responsables devops, deux recetteurs, un UI designer, une UX designeuse, trois product owners, un chef de projet et deux responsables de la conduite du changement. Puis, la prochaine action consiste à trier le reste des données selon trois types : les frustrations, les gains et les besoins. Les gains sont les éléments que chacun.e fait de bien et qu'il faut retenir comme étant un élément positif ou efficace dans le quotidien. En suivant cette méthodologie, j'étais en train de fabriquer l'expérience map(2) de mes collègues ainsi que les fiches personas(3). Les personas représentent un ensemble d'archétypes d'utilisateurs qui ont des besoins et attentes similaires. Finalement, cette étape d'analyse se conclut avec la visualisation de l'ensemble des "pains points" c'est-à-dire les zones douloureuses dans le

(2) Expérience map :

"L'expérience Map, parcours utilisateur ou carte d'expérience, est une synthèse visuelle de l'expérience d'un utilisateur, sa cartographie. La carte d'Expérience permet de comprendre le comportement d'un utilisateur ou d'un client. Cette méthode aide à identifier les points critiques, à engager l'équipe autour d'un projet et à saisir les opportunités d'amélioration ou d'innovation d'un produit ou service. [...] La carte d'expérience ou parcours utilisateur est un moyen de se mettre à la place de l'utilisateur et de ses expériences. Il s'agit d'une représentation graphique et holistique des actions, pensées et émotions des utilisateurs durant la réalisation d'une tâche, ou pour atteindre un objectif." Usabilis

(3) Personas : "Les personas sont des personnes fictives utilisées dans le développement de logiciels informatiques. Il s'agit d'archétypes d'utilisateurs possibles de l'application développée auxquels les concepteurs pourront se référer lors de la conception de l'interface." Wikipédia - Persona Ergonomique

quotidien des utilisateurs. Il ne faut surtout pas parler de solutions à ce stade de la recherche. Nous nous intéressons uniquement aux problèmes et aux opportunités.

Prenons Cédric qui est développeur fullstack. Durant une discussion à la pause, il m'expliquait à quel point le manque de connaissances sur le métier de magasinier sous-traitant le dérangeait au quotidien. Il lui arrivait de chiffrer une user story maladroitement ou tout simplement d'avoir la sensation de "travailler bêtement". Recetteuse depuis pas mal d'années, Anne-Laure se plaignait souvent que les directives n'étaient pas claires. Un jour nous avions trop de budgets et le lendemain il était gelé. Aussi, il lui arrivait de se sentir stressée par les délais qui étaient imposés par nos commanditaires, particulièrement lorsqu'il fallait mettre en production une nouvelle version de l'application. Pourtant les décideurs ne prenaient pas la peine de venir à nos démonstrations. Elle ne voyait aucune reconnaissance. **Figure 2**

La phase de conception (= idéation + génération)

J'étais embarquée dans une sacrée aventure et il m'était impossible d'abandonner le chantier maintenant ! Il me fallait agir, ne serait-ce que pour mettre en évidence les choses qui n'allaient pas très bien dans notre équipe. Selon les sujets, je procédais de différentes manières. Les problèmes observés avaient des degrés de faisabilité qui pouvaient varier allant du : "malheureusement ça ne changera jamais" à "mais pourquoi ne pas l'avoir dit plus tôt". Dès que nous touchons aux sciences humaines, il faut garder à l'œil les règles de déontologie ! J'avais laissé chaque personne décider librement de ce qu'elle souhaitait faire. Ainsi, j'avais organisé des entrevues avec la ou les personnes concernées par un même problème. Puis, on décidait ensemble de l'atelier d'idéation que nous souhaitions mener pour résoudre le problème. Bien sûr je n'aurais jamais pu tout résoudre seule, je préférais faire appel à notre intelligence collective pour générer une solution. Cela me demandait parfois de la diplomatie ou encore du courage pour mettre les pieds dans le plat. En suivant cette logique, nous étions dans cette fameuse étape de conception que j'avais l'habitude de mettre en œuvre pour construire une IHM.

L'étape de conception est le moment où on génère des idées,

on cerne au mieux le périmètre du projet et où on concrétise des idées abstraites. Pour ce faire, nous organisons tout un tas d'ateliers bien réputés : storyboard, design thinking, crazy eight, speed sketching, brainstorming, etc. Ce que nous cherchons c'est la divergence dans les idées. Puis nous finissons toujours par trouver un terrain d'entente pour ainsi finir sur la convergence d'une proposition unique. Il faut parfois un certain temps pour que tout le monde s'aligne sur une même destination finale. C'est une étape itérative, du moins tant qu'aucune solution n'a été retenue. L'essentiel c'est de mettre les bonnes personnes autour de la table pour que ça fonctionne. Les participants peuvent ressentir de la frustration en voyant leurs idées écartées, ce qui peut provoquer de la réticence ou le sentiment d'exclusion par la suite. Donc un petit conseil n'essaye pas de faire des concessions arbitraires, mais plutôt des compromis bien argumentés lors de la prise de décision.

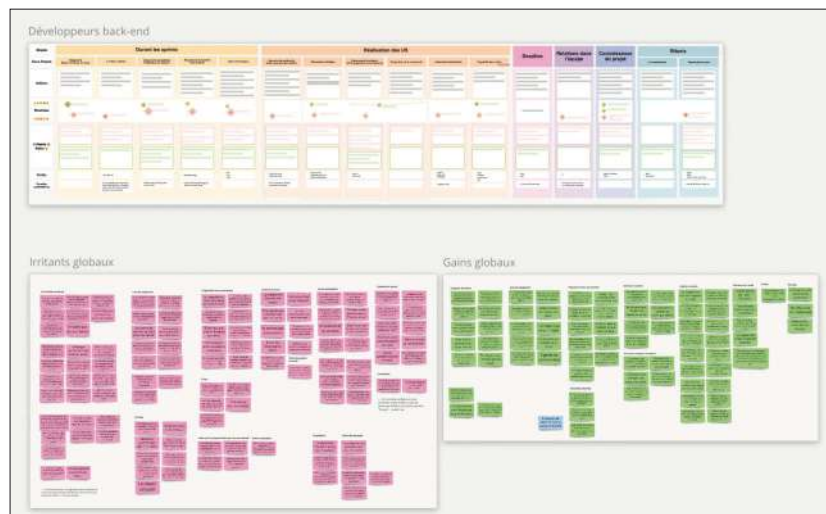
Reprenons notre cher Cédric avec son manque de connaissances métiers. Après discussions, j'ai organisé un temps d'échanges ouvert à tous. C'était comme un forum des métiers. J'ai raconté mes immersions terrains, j'ai montré des photos, j'ai expliqué ce que les professionnels en gestion de stock chez mon client faisaient au quotidien dans leur travail et comment ils utilisaient l'application mobile. Au risque que ce ne soit pas suffisant, nous avons également fait des séances en groupe pour tester l'application. Les personnes qui connaissaient le mieux pouvaient répondre aux différentes questions : "pourquoi ce composant a été fait ainsi ? Pourquoi a-t-il été positionné sur cette page plutôt que l'autre ?" "À quoi sert ce bouton d'exportation ?". Une autre idée très intéressante était ressortie de l'atelier et pourtant elle n'a pas été retenue : il s'agissait de convier les personnes qui le souhaitaient à réaliser une immersion sur le terrain. Une idée très efficace que je recommande à tous tes. Souviens-toi, Anne-Laure trouvait que les directives n'étaient pas claires. Après des discussions impliquant le chef de projet, les product owners et la conduite du changement, nous avons retenu l'idée d'inviter le client dans nos locaux. L'objectif était de créer un lien plus humanisant et réaliste. Pendant que l'équipe montrait l'état d'avancement du projet et les difficultés qu'elle rencontrait chaque jour, le client pouvait expliquer plus clairement quels étaient les enjeux stratégiques et politiques. Une autre solution fût gardée, celle d'organiser chaque semaine un weekly pour partager toutes les informations montantes et descendantes entre l'équipe et les décideurs. Plus de transparence devenait indispensable.

Figure 3

La phase d'évaluation

La machine était enclenchée et les changements opéraient petit à petit. Après quelques jours de repos nécessaire pour prendre du recul, je revenais finalement craintive. Je m'en voulais de ne pas me sentir comblée par le travail accompli. J'avais besoin d'être rassurée, d'entendre le ressenti de mes collègues sur tous ces changements. Était-ce une bonne idée d'avoir secoué la fourmière ? Avons-nous bien fait d'investir de l'argent et de l'énergie dans cette mascarade ? J'avais peur de constater un échec. Il ne faut pas croire, ça arrive bien

Figure 2
Exemple d'une
expérience map résultant
des irritants et des gains
observés durant les
phases d'observation.



souvent. J'ai déjà vécu le boycott des utilisateurs sur notre application mobile. Il n'y a rien de plus déstabilisant pour une UX designeuse que de voir la satisfaction de ses utilisateurs proche de sous terre. Je devais en avoir le cœur net. Il me restait au moins une chose à faire : l'évaluation des solutions.

La phase d'évaluation consiste à tester la qualité des solutions produites en phase de conception auprès des utilisateurs. Les informations recueillies permettent de valider les solutions ou de proposer des recommandations d'amélioration. Comme la seconde étape, celle-ci se veut itérative. Les cycles se terminent lorsque le produit atteint le niveau de qualité attendu par les concepteurs. Pour ce faire il existe des ateliers comme les tests utilisateurs, l'échelle d'utilisabilité, la courbe d'évaluation UX, le test des 5 secondes, etc. Toutes ces méthodes permettent d'évaluer des interfaces numériques, mais ne s'appliquent pas parfaitement à notre situation. Dans notre cas, nous devons nous assurer que les attentes soient satisfaites.

Pour chaque idée mise en œuvre, nous avons trouvé une méthode d'évaluation adaptée. Par exemple, pour Cédric, nous avons mis en place un test de connaissances. C'était un quizz que nous avons fait une fois par mois autour des sujets métier du moment. Aussi nous avons réalisé un sondage de satisfaction pour laisser les personnes comme Cédric s'exprimer librement. Les résultats étaient très prometteurs. Nous nous sommes rendu compte que les développeurs posaient des questions de plus en plus pertinentes. Et ils s'impliquaient davantage pendant les phases de réflexion avant le développement. Enfin, pour ce qui était d'avoir des directives plus claires, il suffisait de voir le sérieux et la régularité qui ressortait de ces points de synchronisation chaque semaine. Suite à certains feed-back non satisfaisants, nous avons travaillé sur une solution supplémentaire. Nous avons installé une roadmap en papier sur le mur pour avoir une vision globale de l'évolution du projet. Enfin, pour clôturer le tout, nous avons mené des interviews, tout comme durant la phase exploratoire. Ainsi nous disposions de deux captures d'écrans de la vision de notre équipe à deux moments différents. Nous avons un avant et un après, il suffisait de comparer les deux. Je ne vais pas te cacher que le meilleur moment pour évaluer notre fonctionnement et notre satisfaction était durant les rétrospectives. Les retours étaient très positifs bien qu'il y avait encore des problèmes qui surgissaient de nulle part. Cette méthode devait encore prouver son efficacité pour obtenir la confiance de l'équipe. En attendant, nous avons tous compris que nous devions rester à l'écoute et attentif au bien-être des uns et des autres.

Conclusion

C'est seulement maintenant, plusieurs mois après avoir vécu cette expérience, que je me suis aperçue qu'il s'agissait d'une mission humanitaire(4). J'ai choisi bénévolement et volontairement d'aider ces gens, d'essayer de résoudre des problèmes humains complexes. D'ailleurs, il n'y avait pas de solutions vraies ou fausses, il fallait simplement décider si nous voulions des solutions bonnes ou mauvaises. La nuance est importante, car elle m'a fait m'interroger sur mon rôle

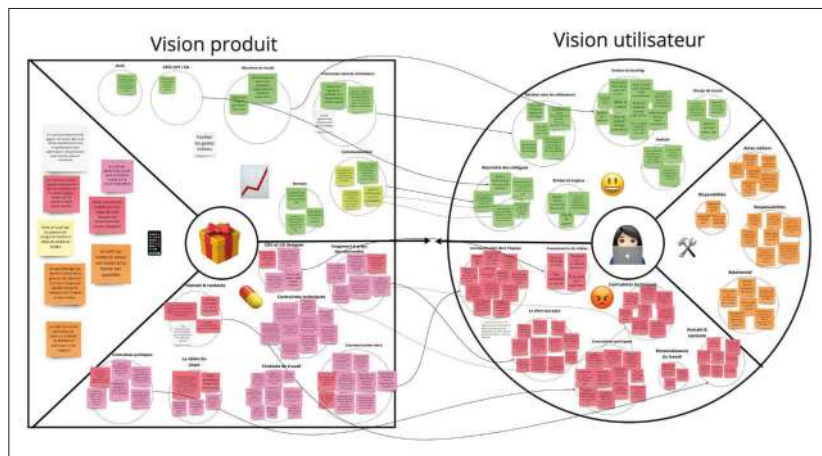


Figure 3

La Value Proposition Canvas permet de créer de la valeur en partant des problèmes observés et en trouvant des solutions numériques ou non.

éthique au sein du projet. Il fallait tenir compte de tout ce qui entourait le projet pour résoudre l'équation, soit ce qui ressortait des strates du client et des utilisateurs de l'application mobile. À la suite de cela j'ai découvert que certaines méthodologies, comme le design systémique(5), étudient la relation entre tous les éléments d'un ensemble et cela permet de prendre conscience des problèmes qui gravitent autour. Ces problèmes peuvent être de nature très différente : il peut s'agir d'enjeux sociaux, écologiques, environnementaux ou politiques. Ce domaine d'expertise est encore très récent c'est pourquoi il est difficile de trouver des exemples concrets, toutefois ce raisonnement est inspirant.

Pourquoi ne pas avoir parlé des méthodes agiles, me dirais-tu ? Effectivement l'agilité est très répandue au sein des entreprises travaillant dans le numérique. Cette philosophie a aussi inspiré bien au-delà du contexte professionnel pouvant aller jusqu'au monde scolaire ou la vie familiale. L'agilité est l'huile qui permet au moteur de ne pas se crispier. Ça a pour objectif d'assurer le bon déroulement du projet, de faire dialoguer les humains entre eux et surtout de tout faire pour que le produit final soit réalisé. Alors que dans mon cas, l'UX design est tout ce qui est autour du moteur. Il s'agit de faire en sorte que les éléments qui composent la voiture soient cohérents, indispensables et en bon état pour que le moteur fonctionne. Finalement cette expérimentation m'a permis de comprendre ce qui différencie l'univers Agile de celui du design d'expérience. Il faudrait probablement un autre article entier pour parler de tout cela !

Selon ma vision, un agiliste perçoit le monde d'une manière qui complète celle vue par une UX designeuse. La designeuse porte son attention sur des détails, lit entre les lignes et utilise l'empathie comme une force. Une personne travaillant dans la recherche utilisateur développe de nombreuses capacités émotionnelles : l'habitude d'être dans une posture d'observation discrète, d'être en position diplomatique, elle évite les biais cognitifs, communique avec toutes les personnes de l'équipe ou encore à une vision micro et macro dans le projet. Cette personne va savoir employer les bons mots pour dire ce qu'elle ressent, qui est différent de ce que l'on pense. L'un est factuel à la différence de l'autre. Cette intelligence émotionnelle s'alimente à chaque fois que nous pratiquons notre métier et cela nous permet de comprendre la complexité des sciences humaines et cognitives.

(4) Définition du terme humanitaire :

<http://www.aidehumanitaire.org/humanitaire/>

(5) Définition du Design Systémique de Wikipédia :

https://en.wikipedia.org/wiki/Systemic_design



Michelle Avomo

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.



Romy Alula

Consultante en développement Web chez Codeworks. Membre active de Ladies of Code et Duchess France, elle y accompagne de jeunes dev en tant que marraine. Sur des sujets comme le Craft, la préparation d'entretien technique ou encore la négociation. Depuis mars 2021, elle anime le hands-on sur le TDD, chez les Ladies of code. Dès qu'elle a du temps libre, elle joue du piano et encourage ses enfants à exprimer leur créativité. @Goumies

Approval testing : l'art de sécuriser la refonte de son legacy

Les tests sont le premier client de notre code. Ils nous donnent un feedback quasi immédiat sur le comportement du système testé face à un bug ou une nouvelle fonctionnalité.

Sans tests, la développeuse est privée de ce précieux premier feedback et est exposée à deux risques :

- 1- Casser une fonctionnalité clé de la production
- 2- Casser une fonctionnalité clé de la production **au pire moment** (ex: veille de démo)

Quel que soit son niveau d'expérience, la crainte légitime de causer des régressions est redoublée quand on connaît très peu le code legacy à corriger (ou à faire évoluer) et que ce dernier comporte peu ou pas de tests. Il existe depuis quelques années déjà des techniques de tests éprouvées pouvant aider à explorer un code non testé en un minimum de temps.

Dans le cadre d'un BBL, nous avons exploré la façon de sécuriser un code JS ne contenant quasiment pas de tests à l'aide de la technique dite du « Golden Master ».

Principes

Contrairement aux tests pré-production "classiques" (unitaires, intégration, end-to-end) qui assurent de la pérennité d'une fonctionnalité donnée, les tests "approvals" visent à faire un état des lieux du code. Toute la puissance de cette technique réside dans le fait d'écrire un minimum test permettant de produire un résultat (textuel ou visuel) montrant les changements enregistrés.

Il n'y pas de préoccupations liées aux dépendances ni aux sources de données dans ce type de test et encore moins de prise en compte de l'implémentation.

Par ailleurs, parce qu'il montre l'état des lieux d'un code donné à un instant précis, ce type de test est nécessairement éphémère, car, la correction d'un bug par exemple change fatalement le comportement initial du logiciel. Il n'y a pas de logique de préservation d'un comportement du code.

Mais, à la manière d'un gestionnaire de version, un test "Approval" montre à chaque exécution, les différences entre le résultat d'une exécution initiale et les changements intervenus. La développeuse choisit quel changement approuver. Les tests "Approvals" sont ainsi les seuls types de tests qui doivent être supprimés dès qu'une nouvelle fonctionnalité est implémentée ou un bug est corrigé.

Par ailleurs, il est courant d'associer l'exécution d'un test "Approval" avec un outil de couverture de code afin de mieux comprendre quelle portion de code est couverte par quels jeux de tests. Très pratique dans une phase d'accoutumance au code legacy, nous montrons dans le point suivant comment démarrer avec l'Approval testing.

Usages

Il existe depuis plusieurs années des outils clefs en main permettant de faire de l'Approval testing. La librairie de référence s'appelle d'ailleurs "ApprovalTest" et a été développée par [Llewelyn Falco](#).

Dans un autre registre, Jest - le framework de test de Facebook - a popularisé la technique de l'approval testing par un brillant recyclage sous le terme de "snapshot testing". Cependant, les outils, aussi utiles soient-ils, ont tendance à mystifier des pratiques pourtant accessibles.

C'est pour permettre à qui veut de comprendre le concept derrière le snapshot testing de Jest que nous avons opté, dans le cadre de cet article, pour une implémentation manuelle.

Génération manuelle des entrées/sorties

Pour faire un état des lieux pertinent de son code, il est conseillé de procéder en 3 étapes pour toutes fonctionnalités à tester.

- Passez des paramètres d'entrées que vous maîtrisez à l'application et exécutez les tests progressivement puis, enregistrez la sortie complète dans un fichier.
- Créez un test automatisant l'injection des mêmes données dans le système. Le test doit être capable de capturer la sortie du programme (devenue prédictible) et de le comparer avec la sortie de l'étape 1 ci-dessus.
- Mesurez la couverture de test. Répéter l'opération jusqu'à ce que la couverture de test soit à près de 100%.

Et voilà !

LE CAS PRATIQUE

Chez CodeWorks, nous avons créé un kata maison faisant office de legacy sur lequel il est demandé d'assurer la refonte, avant de pouvoir ajouter de nouvelles fonctionnalités. Par exemple l'évolution du calcul du score. Pour une question ouverte, la personne qui valide les réponses du candidat doit pouvoir ajouter un bonus de 0,5 point, si la réponse est très satisfaisante.

Vous pouvez retrouver ce kata sur notre GitHub : <https://github.com/CodeWorksFrance/kata-technical-workshop>

Le fonctionnement du code de production Figure 1

Il existe une liste de questions à poser à un candidat.



Chaque question est liée à un langage et à un niveau de difficulté prédéfinis.

```
<pre>
[ {
  label: 'JavaScript',
  questions: [
    {
      label: 'How to find the length of the string?',
      answer: 'String.length',
      difficulty: 1
    }
  ]
},
{
  label: 'JavaScript',
  questions: [
    {
      label: 'How to return a random number between 0 and 1?',
      answer: 'Math.random()',
      difficulty: 1
    }
  ]
}
]
]>
</pre>
```

Suivant le profil du candidat, CodeTest affiche les questions liées aux langages choisis. Lorsque le candidat a fini de répondre aux questions, un.e CodeWorker doit évaluer les réponses du candidat. La personne qui procède à l'évaluation n'est pas forcément experte sur les technos sélectionnées. À ce jour, il n'existe qu'un test unique pour le cas où la personne refuse de répondre aux questions.

Présentation de la stack technique

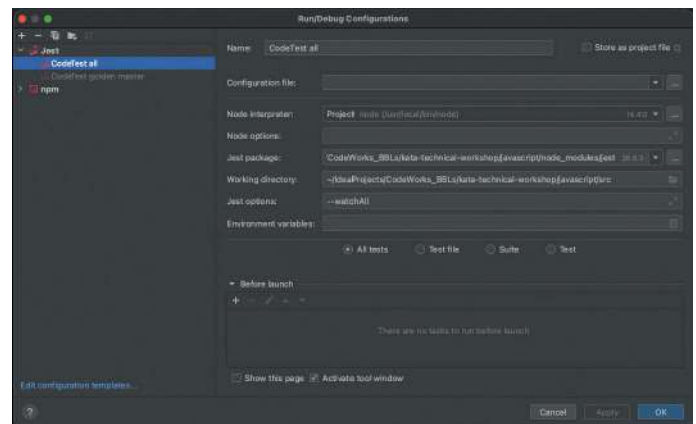
Pour construire ce kata, nous avons choisi les technologies suivantes :

- une interface en ligne de commandes réalisée avec Node
- des questions stockées dans un fichier json
- le test runner Jest
- la dépendance prompt-sync pour poser des questions et retourner la réponse saisie. La méthode prompt appartenant à la Web API. Nous voulions avoir le même comportement sous Node.

Notez que nous utilisons Jest comme test runner. Vous pouvez vous demander pourquoi nous n'utilisons tout simplement pas les snapshots de Jest. C'est un choix délibéré. Le snapshot est un exemple de "golden master" (<https://alisterb-cott.com/2020/10/28/api-approval-testing-using-jest/>). Nous avons choisi d'illustrer la redirection et la sauvegarde de la sortie attendue. Plutôt que de faire une démonstration d'un outil. Ainsi, nous avons la main sur toute l'implémentation de notre golden master. C'est d'ailleurs de cette façon que, moi, Romy, je l'ai personnellement découvert. À l'époque, c'était dans l'écosystème Java.

Pour lancer tous les tests en continu, sous IntelliJ ou Webstorm, vous pouvez exécuter la commande `npm start` dans votre terminal. Ou utiliser la configuration suivante :

passer `--watchAll` dans les Jest options et sélectionnez le bouton radio `All tests`



Configuration d'exécution automatique de tous les tests

Au lancement de l'application

La commande « npm start » permet de lancer l'application.

Par défaut, le profil du candidat est développeur.se Java.

CodeTest nous souhaite, ensuite la bienvenue et indique que nous aurons 8 questions en Java.

À la question "Are you ready ?", taper "y" et "Entrée" fait défiler les questions. Taper tout autre caractère et "Entrée" arrête l'application, après indication du score de 0 point.

```
> kata-technical-workshop-js@1.0.0 start
> node --experimental-json-modules src/index.js

Adding SQL in categories
Adding Toto as candidate
Welcome to the interview game. You'll have 8 questions on Java
Are you ready? Press y and Enter to start.

***** Response from: Toto *****

The candidate as a total of 0 points.
```

La génération de la sortie du code de production

```
const PRODUCTION_OUTPUT = []

const generateProductionOutput = () => {
  TECHNICAL_WORKSHOP.addCat = jest.fn().implementation(() => PRODUCTION_OUTPUT.push('Adding SQL in categories\n'))
  TECHNICAL_WORKSHOP.addCandidate = jest.fn().implementation(() => {
    PRODUCTION_OUTPUT.push('Adding Toto as candidate\n')
  })
  TECHNICAL_WORKSHOP.runCodeTest = jest.fn().implementation(() => {
    PRODUCTION_OUTPUT.push('Welcome to the interview game.\n')
    PRODUCTION_OUTPUT.push('You'll have 8 questions on Java\n')
    PRODUCTION_OUTPUT.push('Are you ready? Press y and Enter to start.\n')
    PRODUCTION_OUTPUT.push('***** Response from: Toto *****\n\n')
  })

  console.log = jest.fn().implementation(() => PRODUCTION_OUTPUT.push('The candidate as a total of 0 points.\n'))
  runCodeTest()

  return PRODUCTION_OUTPUT
}

module.exports = generateProductionOutput
```

Sortie attendue en production

La génération de la sortie à la demande

Dans le module chargé de générer la sortie à la volée, nous utiliserons des variables dynamiques, en lieu et place des chaînes de caractères en dur.


```

11 TECHNICAL_WORKSHOP.addCandidate = jest.fn(implementation, (firstName) => {
12   thatFirstName = firstName;
13   ON_DEMAND_OUTPUT.push('Adding ${firstName} as candidate\n');
14 });
15 TECHNICAL_WORKSHOP.runCodeTest = jest.fn(implementation, (category) => {
16   ON_DEMAND_OUTPUT.push('Welcome to the interview game.\n');
17   ON_DEMAND_OUTPUT.push('You'll have 8 questions on ${category}\n');
18   ON_DEMAND_OUTPUT.push('Are you ready? Press y and Enter to start.\n');
19   ON_DEMAND_OUTPUT.push(
20     '***** Response from: ${thatFirstName} *****\n\n'
21   );
22   return 0.0;
23 });
24
25 console.log = jest.fn(implementation, scoreDisplay => ON_DEMAND_OUTPUT.push(scoreDisplay));
26 runCodeTest();
27
28 return ON_DEMAND_OUTPUT;
29 };
30
31 module.exports = generateOnDemandOutput;
32

```

Sortie générée à la demande

Notez que dans les deux cas, nous appelons le code de production, avec l'instruction `runCodeTest()`.

La comparaison des deux sorties

```

1 const generateProductionOutput = require('./utils/test/generateProductionOutput');
2 const generateOnDemandOutput = require('./utils/test/generateOnDemandOutput');
3
4 describe('name: "CodeTest"', in function () {
5   describe('name: "golden master"', in function () {
6     it('name: "should compare the expected output, given another key than "y" when the candidate is asked if ready"', in function () {
7       const ON_DEMAND_OUTPUT_DATA = generateOnDemandOutput().join('\n');
8       const PRODUCTION_OUTPUT_DATA = generateProductionOutput().join('\n');
9       expect(PRODUCTION_OUTPUT_DATA).toBeStrictEqual(ON_DEMAND_OUTPUT_DATA);
10     });
11   });
12 });
13

```

Implémentation du golden master

C'est le seul test que nous rédigeons dans ce fichier. Son unique responsabilité consiste à comparer la sortie en production et celle qui est générée à la volée.

La sécurisation du code de production

Pour commencer, nous pouvons penser le code de production comme une boîte noire.

Le point d'entrée a pour unique responsabilité le lancement de CodeTest. Une fois son exécution finie, le process Node.js est terminé.

```

1 const runCodeTest = require('./codeTestRunner.js');
2
3 runCodeTest()
4 process.exit()
5

```

Point d'entrée de l'application

```

1 const TECHNICAL_WORKSHOP = require('./technicalWorkshop.js')
2
3 function runCodeTest(){
4   TECHNICAL_WORKSHOP.addCat('label: 'SQL')
5   TECHNICAL_WORKSHOP.addCandidate('firstName: 'Toto', 'lastName: 'Titi', 'email: 'titi@mail.fr')
6
7   const SCORE = TECHNICAL_WORKSHOP.runCodeTest('category: 'Java')
8   console.log('The candidate as a total of ${SCORE} points.')
9 }
10
11 module.exports = runCodeTest
12

```

CodeTestRunner

Maintenant que tout est en place, nous allons casser le test, pour nous assurer qu'il joue bien son rôle.

Quand je remplace 'SQL' par 'Java', ligne 4 dans codeTestRunner.js, le test échoue : la valeur attendue contient 'Java' et non 'SQL', qui est contenue dans la sortie en production.

```

error: expect(received).toBeStrictEqual(expected) // deep equality
Expected: ""
Received: "0.0"

```

Echec du test, après modification du code de production

```

1 const promptSync = require('prompt-sync')
2 const prompt = promptSync( config: {} )
3
4 const Q = require('./questions.json')
5
6 let CAN = { firstName: '' }
7 const CAT = []
8
9 const TECHNICAL_WORKSHOP = { cat: CAT... }
10
11 module.exports = TECHNICAL_WORKSHOP

```

TechnicalWorkshop

Le golden master va pouvoir pleinement entrer en action avec les tests de caractérisation. Ces tests exploratoires vont venir documenter le code.

Laissons parler le code

Place à l'exploration. Le golden master sécurise la sortie attendue d'une entrée donnée. Quand les tests de caractérisation, eux, viennent révéler le comportement du code de production. À mesure que nous rédigeons des tests de caractérisation, le résultat actuel nous donne le résultat attendu, qui fera passer le test en cours. Un premier test de caractérisation pourrait nous révéler ce qui se passe quand `TECHNICAL_WORKSHOP.addCat('SQL')` est appelée ligne 4 dans codeTestRunner.js.

```

1 const TECHNICAL_WORKSHOP = require('./technicalWorkshop.js');
2
3 describe( name: 'CodeTest', in function(){
4   describe( name: 'technical workshop', in function(){
5     it( name: 'should ???', in function(){
6       expect(TECHNICAL_WORKSHOP.addCat( label: 'SQL')).toBeStrictEqual( expected: '' )
7     });
8   });
9 }

```

Echec du test

```

error: expect(received).toBeStrictEqual(expected) // deep equality
Expected: ""
Received: undefined

```

Echec du test, révélation du retour

Le compilateur et le résultat du test nous indiquent que cette fonction ne renvoie rien. Nous mettons à jour notre test avec cette nouvelle information.

```

1 const TECHNICAL_WORKSHOP = require('./technicalWorkshop.js');
2
3 describe( name: 'CodeTest', in function(){
4   describe( name: 'technical workshop', in function(){
5     it( name: 'should return undefined', in function(){
6       expect(TECHNICAL_WORKSHOP.addCat( label: 'SQL')).toBeStrictEqual( expected: undefined )
7     });
8   });
9 }

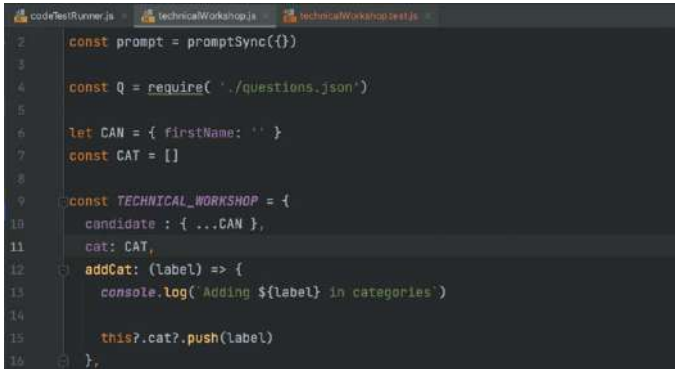
```

Succès du test



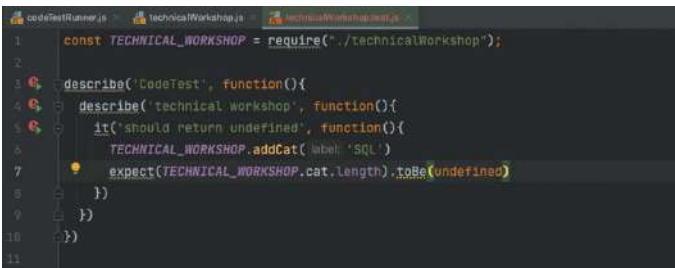
Succès du test, révélation du retour

Les tests passent. Par la même occasion, nous retrouvons le log 'Adding SQL in categories', comme au lancement de l'application. Il ne reste plus qu'à s'occuper de l'avertissement du compilateur. La fonction ne retournant rien, le compilateur nous avertit qu'un tel test n'a pas lieu d'être.

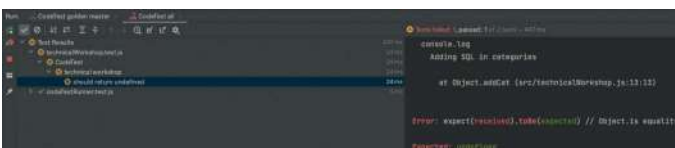


Dans technicalWorkshop.js

Notez qu'après l'affichage du log, un tableau 'cat' est alimenté. Il convient donc mieux de vérifier la longueur de 'cat'.



Échec du test, après évolution



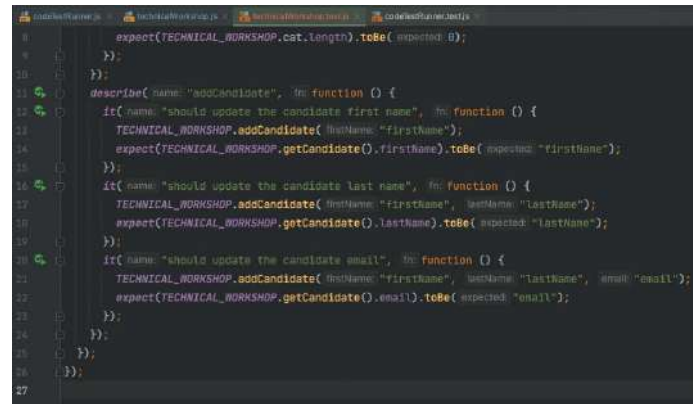
Échec du test, après évolution

Étonnement, alors qu'un label est ajouté ligne 15 dans 'technicalWorkshop.js', la longueur du tableau est ou reste égale à 0.

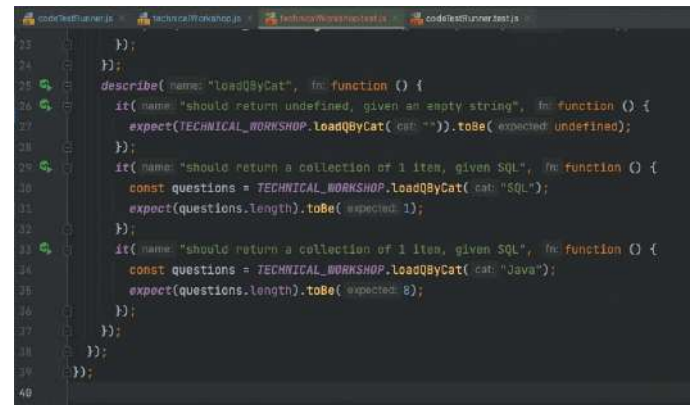


Succès du test, après évolution

Poursuivons l'exploration, pour en apprendre un peu plus.



Succès du test, après évolution



Succès du test, après évolution

L'ensemble des tests, en combinaison avec le golden master, nous permet de refactorer avec assurance.



Début refactoring, après renommage 'loadQuestionsByCategory'

Nous vous invitons maintenant à poursuivre l'exploration. Le Métier aimerait revoir le calcul du score. Lorsque le/la candidat/e donne 3 bonnes réponses successives, le score est augmenté d'un bonus de 3 points. Vous avez maintenant toutes les connaissances pour mener à bien ce refactoring et mettre le nouveau calcul du score en production.

Conclusion

À travers CodeTest, nous avons pu aborder une première approche des concepts d'"approval testing" et tests de caractérisation. L'implémentation manuelle permet de rediriger puis stocker les sorties. Leur comparaison en continu assure le résultat de l'exécution de l'application. Vous avez découvert les tests de caractérisation, révélateurs des spécifications de l'application en production. N'hésitez pas à poursuivre l'exercice pour vous entraîner. Et livrer la nouvelle évolution, dans un cadre sécurisé et reproductible.



Armelle YOUNBI

Développeuse full-stack
reconvertie dans le
monde de la
programmation .Net/C#
Angular depuis 3 ans.
J'aime apprendre de
nouvelles choses.

Comprendre RXJS (les observables et les opérateurs)

RXJS (Reactive Extension for Javascript), les extensions réactives ont été inventées par l'équipe de programmation cloud de Microsoft en 2011 en tant qu'un sous produit de Volta (ensemble d'outils de développement expérimental pour la création d'applications web multi tiers). L'implémentation initiale de Rx était pour .NET Framework et a été publiée le 21 juin 2011. Plus tard, l'équipe a commencé l'implémentation de Rx pour d'autres plateformes, y compris JavaScript et C++. La technologie est sortie en open source fin 2012.

La plupart des applications modernes de nos jours ont besoin de réponses ultra-rapides et des capacités de traiter en même temps des données provenant de différentes sources sans rien manquer. La programmation réactive est une façon naturelle et plus simple de penser au code asynchrone. Il est difficile de parler de RXJS sans mentionner les observables et encore plus de parler des observables sans mentionner les 'javascript promises'. Les 'promises' ont été introduites avec ES (2015) comme une façon plus propre de concevoir des callback fonctions et du code asynchrone. RXJS, lui, est venu ensuite pour mieux gérer et écrire du code propre autour de la programmation asynchrone javascript, ainsi qu'offrir, avec de nouveaux opérateurs, une façon de mieux itérer sur les flux de données.

Que signifie RXJS ?

RXJS (Reactive extensions for JavaScript) est une bibliothèque pour composer des programmes asynchrones (tout comme les AsyncPipe) et basés sur des événements en utilisant des séquences observables.

Avec RXJS les concepts suivants permettent de manipuler les tâches asynchrones :

- **Observable** : Un observable est une fonction retournant un flux de valeurs à un observateur de manière synchrone ou asynchrone. Un observable s'exécute s'il y a un observer (observateur) et un abonnement.
- **Observer** : Un Observer est un objet capable d'écouter toutes les notifications d'un observable : il contient 3 callbacks (next, error et complete).
- **Subscription** : Pour exécuter l'observable, nous devons y souscrire (s'abonner) comme pour un événement et on se désinscrit (désabonne) avec 'unsubscribe'.
- **Operators** : Un opérateur est une fonction pure qui prend en entrée un observable et retourne un observable.
- **Subject** : Un subject a à la fois le rôle d'observer et d'observable et il autorise la souscription de plusieurs observer.
- **Schedulers** : Un scheduler contrôle quand l'exécution doit débuter ou être notifié.

Dans cet article nous allons nous concentrer sur les **observables** et quelques **opérateurs**, car je pense que ce sont les sujets les plus traités avec RXJS.

Les Observables

RXJS tire ses fondations du pattern observer. Ce patron de conception décrit la relation 1-n entre plusieurs objets, de telle sorte que dès que l'état d'un des objets change les autres en sont notifiés et modifiés automatiquement.

Un observable met en relation un producteur et un récepteur, il crée l'observateur (observer) qui va écouter le producteur. Un 'observer' est le récepteur, aussi appeler l'abonné. L'observer dispose des méthodes :

- **'next'** : Il permet à l'observer de retourner la ou les valeurs émises par l'observable.
- **'complete'** : Il notifie l'observer que l'observable a terminé d'envoyer des notifications. Si l'appel à la méthode 'complete', n'est pas fait, un observable infini est produit et nous risquons des fuites mémoire si l'on oublie de se désabonner de l'observable.
- **'error'** : Il notifie l'observer que l'observable a rencontré une erreur.

Un 'observer' peut se connecter à un observable en s'y inscrivant. Le résultat est connu sous le nom 'subscription'.

Les observables sont déclaratifs : on déclare une fonction mais le traitement ne s'exécutera pas tant que le consommateur ne s'y abonnera pas. L'abonné 'observer' reçoit alors des notifications jusqu'à ce que le traitement se termine ou jusqu'à ce qu'il se désabonne.

Exemple :

```
import { Observable } from 'rxjs';
//création de l'observable
const data$ = new Observable(observer => {

  observer.next(1);
  observer.next(2);
  observer.next(3);
  observer.complete();

});
//souscription à l'observable
data$.subscribe({
```

Références :

Reactive Programming with RxJS: Untangle Your Asynchronous JavaScript Code,
<https://medium.com/@mamodsayed/everything-you-need-to-know-about-rxjs-6-740f1fde09a7>
<https://en.wikipedia.org/wiki/ReactiveX>

```
//retourne les valeurs émises par next
next: value => console.log("Valeur retournée: " + value),
//notification d'erreur
error: err => console.error("Une erreur est survenue: " + err),
//notification de fin d'exécution.
complete: () => console.log("Le traitement est terminé")
}).unsubscribe();// désabonnement à l'observable.
```

Un 'observer' peut définir plusieurs combinaisons d'événements, mais si aucun type n'est défini alors par défaut aucun n'est géré. Il est important de se désinscrire des observables car cela permet de libérer la mémoire, d'éviter les fuites de mémoires, d'éviter la consommation inutile de CPU et les effets de bord associés aux observables.

Les Opérateurs

Un Opérateur est une fonction qui prend en entrée un observable, exécute une logique sur les valeurs retournées via l'observable et crée un nouvel observable avec ces valeurs, sans modifier l'observable d'origine. La méthode `pipe()` est une méthode qui crée un tuyau pour relier les opérateurs entre eux. Ce tuyau permet de combiner plusieurs fonctions en une seule. Elle prend en arguments les fonctions qu'on souhaite combiner et renvoie une nouvelle fonction qui, lorsqu'elle est exécutée, exécute les fonctions composées dans l'ordre.

Il existe plusieurs types d'observable tels que :

- **Opérateurs de combinaisons** : qui permettent de réunir les informations venant de plusieurs observables (exemple : `concat`, `combineAll`, `endWith`).
- **Opérateurs conditionnels** : pour les cas où l'on veut que le traitement ne se fasse que si la condition est réunie (exemple : `every`, `iif`).
- **Opérateurs de création** : permettent la création d'observables à partir de différentes sources de données (exemple: `create`, `of`, `from`).
- **Opérateurs "error handling"** : permettent la prise en charge d'erreurs et la mise en place de nouvelles tentatives (exemple: `catch` / `catchError`, `retry`).
- **Opérateurs de filtre** : permettent de choisir quand et comment accepter les valeurs retournées par les observables (exemple: `take`, `filter`, `find`).
- **Opérateurs de transformation** : ce sont des fonctions qu'on peut utiliser pour modifier/transformer les valeurs émises par les observables (exemple: `map`, `switchMap`).
- **Opérateurs d'utilité** : ces opérateurs permettent la journalisation, la gestion des notifications et la configuration des planificateurs (exemple: `let`, `repeat`, `delay`).
- **Opérateurs de multidiffusion (multicast)** : avec ces opérateurs contrairement aux observables où chaque souscription relance le traitement, ici une souscription est partagée entre plusieurs souscripteurs (exemple: `publish`, `share`).

Je vais vous présenter quelques-uns des opérateurs les plus utilisés en Angular :

map() : `map()` est l'opérateur le plus utilisé en RXJS. Il permet de créer un nouvel observable à partir de l'observable de base en modifiant tout simplement les valeurs émises.

```
import { from } from 'rxjs';
import { map } from 'rxjs/operators';
```

```
//from() crée un observable à partir d'un tableau de nombre
const source = from([2, 4, 6]);
//multiplication de chaque valeur par 5
const example = source.pipe(map(val => { val*5 }));
const subscribe = example.subscribe(val => console.log(val));
//retourne: 10,20,30
```

```
import { from } from 'rxjs';
import { map } from 'rxjs/operators';

/*retourne ({ nom: 'Armelle', age: 33 },
{ nom: 'Lionel', age: 32 }, { nom: 'Eric', age: 29 });*/
const ListeNomAge = from([
  { nom: 'Armelle', age: 33 },
  { nom: 'Lionel', age: 32 },
  { nom: 'Eric', age: 29 }
]);
//récupère le nom de chaque personne
const example = ListeNomAge.pipe(map(({ nom }) => nom));
const subscribe = example.subscribe(valeur => console.log(valeur));
//retourne: "Armelle", "Lionel", "Eric"
```

catchError() : Elle permet la prise en charge des erreurs pouvant arriver lors de l'exécution d'un observable en retournant un observable ou une erreur.

```
import { from, of } from 'rxjs';
import { map, catchError } from 'rxjs/operators';

const source = from([3, 6, 8]);
//vérifie les nombres modulo de 3
const example = source.pipe(map(val => {
  if (val % 3 === 0) {
    throw "val + ' n'est pas divisible par 3';
  }
  return val;
})),
//of() retourne un observable à partir de plusieurs sources de données
(string, tableau, objet, ...)
catchError(err => of('10', '12', '14'))).subscribe(x => console.log(x));
//retourne: 3,6 => l'erreur se produit et retourne 10,12,14
```

mergeMap() : Ceci est une combinaison de deux opérateurs (`merge` et `map`). Ici lorsqu'on souscrit à l'observable, la fonction `map()` retourne un observable interne. Un abonnement est fait sur cet observable interne qui transmet les valeurs retournées à un autre observable. Lorsque l'observable initial se termine, l'ensemble de la chaîne d'observable s'achève également.

```
import { from, interval, zip } from 'rxjs';
import { map, mergeMap } from 'rxjs/operators';
```

```
/*zip() permet de combiner des observables pour créer
un observable qui retourne des valeurs calculées dans l'ordre
des observables d'entrée. */
```

```
/*interval() crée un observable qui émet des nombres
séquentiel à un intervalle de temps donné. */
```



```

const getCurrentCity = () => {
  return zip(
    from(['Strasbourg', 'Paris', 'Lyon']),
    interval(1000)
  )
  .pipe(map((city) => city));
};

const getTemperature = city => {
  return interval(400)
    .pipe(map(() => 100 / city.length));
};

const currentCityTemperature$ = getCurrentCity()
  .pipe(mergeMap(city => getTemperature(city)));

currentCityTemperature$
  .subscribe(température => console.log(température));
//retourne
// à 1000ms, nous sommes à Strasbourg.
// 10 # à 1400ms, température à Strasbourg.
// 10 # à 1800ms, température à Strasbourg.
// # à 2000ms, nous sommes à Paris.
// 10 # à 2200ms, température à Strasbourg.
// 20 # à 2400ms, température à Paris.
// 10 # à 2600ms, température à Strasbourg.
// 20 # à 2800ms, température à Paris.
// # à 3000ms, nous sommes à Lyon.
// 10 # à 3000ms, température à Strasbourg.
// 20 # à 3200ms, température à Paris.
// 25 # à 3400ms, température à Lyon.
// 10 # à 3400ms, température à Strasbourg.
// 20 # à 3600ms, température à Paris.
// 25 # à 3800ms, température à Lyon.
// 10 # à 3800ms, température à Strasbourg.
// 20 # à 4000ms, température à Paris.
// 25 # à 4200ms, température à Lyon.
// 10 # à 4200ms, température à Strasbourg.
// ...

```

switchMap() : Tout comme l'opérateur mergeMap(), lorsque l'on souscrit à l'observable, la fonction map() retourne un observable interne et ensuite un abonnement est fait sur cet observable interne. Il crée un nouvel observable intérieur pour chaque valeur qu'il reçoit de l'observable source. Chaque fois qu'il crée un nouvel observable interne, il se désinscrit de tous les observables internes précédemment créés. Fondamentalement, il passe au plus récent observable en éliminant tous les autres.

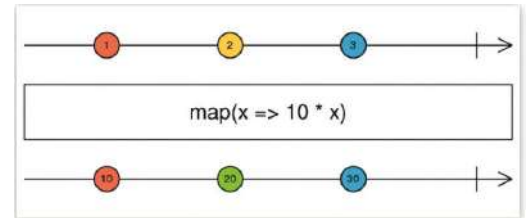
Code complet sur [programmez.com](https://www.programmez.com) & [github](https://github.com)

A la différence du mergeMap ici lorsque l'observable interne émet une nouvelle valeur, le précédent observable est annulé ; raison pour laquelle à partir de 2000 ms on arrête de retourner des valeurs liées à Strasbourg.

Difficultés liés à RXJS

Selon mon expérience la prise en main de RXJS n'est pas évidente surtout avec tous les concepts autour des observables. Pour des besoins de test on associe souvent

RXJS avec le Marble diagram qui est un outil spécifique développé par l'équipe RXJS. Malheureusement, la nécessité d'utiliser des outils supplémentaires rend le processus plus compliqué : la syntaxe des [diagrammes Marble](#) est assez délicate, ce qui nécessite un apprentissage supplémentaire. Exemple de Marble diagram pour un opérateur map() :



Par ailleurs, certains opérateurs RXJS ont le même nom que quelques fonctions importées de bibliothèques différentes ce qui peut provoquer des conflits, ce problème peut être contourné en transférant une des fonctions dans un composant différent et l'appeler dans le composant voulu.

Conclusion

RXJS évolue rapidement, elle possède plus de 2,4 millions de repositories sur GitHub, plus de 16 000 packages NPM, et plus de 22 millions de téléchargements par semaine. Elle peut être utilisée avec d'autres bibliothèques javascript et autres frameworks comme Angular, ReactJS, Vue.js, etc. Comme nous l'avons mentionné plus haut, elle aide dans le traitement efficace des tâches asynchrones et les programmes basés sur les événements. RXJS a cet avantage que les observables ont une "exécution lente" : elles sont déclaratives et ne s'exécutent pas s'il n'y a pas de souscription. On peut aussi parler de "lazy loading". Ceci permet d'écrire un code plus efficace. Les problèmes associés à la perte de mémoire lors des exécutions ont été détectés et traités. RXJS possède une large communauté qui augmente tous les jours, s'entraide et fournit des explications approfondies des concepts, des meilleures pratiques et des exemples pour aider les utilisateurs à améliorer leurs connaissances et compétences théoriques.

Liens

StackOverflow (<http://stackoverflow.com/questions/tagged/rxjs>)

Gitter (<https://gitter.im/Reactive-Extensions/RxJS>)

GitHub (<https://github.com/ReactiveX/rxjs>)

https://www.ru.nl/publish/pages/769526/dorus_peelen.pdf

Reactive Programming with RxJS 5: Untangle Your Asynchronous JavaScript Code

<https://angular.io/guide/observables>

<https://www.thisdot.co/blog/basic-rxjs-operators-and-how-to-use-them>

<https://www.learnrxjs.io/learn-rxjs/operators>

<https://guide-angular.wishtack.io/angular/observables/operateurs/mergemap-and-switchmap>

<https://www.javatpoint.com/advantages-and-disadvantages-of-rxjs>

<https://kruschecompany.com/rxjs-pros-and-cons/>

<https://blog.angular-university.io/rxjs-higher-order-mapping/>

Dehors, le monde est rempli de dragons !

Dans cet article, nous allons parler d'un sujet qui revient dans chaque projet : « la validation », sujet très complexe, même si nous travaillons tous avec au quotidien, c'est souvent difficile de choisir entre plusieurs approches.

Figure 1

La validation est un processus qui consiste à s'assurer que notre code fonctionnera sur des données propres, correctes et utiles. Il s'agit de vérifier l'exactitude des données. Toutes les données d'entrée doivent être validées, peu importe d'où elles viennent. La validation ce n'est pas la validation des règles métier, il s'agit de s'assurer que les données qui entrent dans notre système ont un certain niveau d'intégrité qui est cohérent en interne. Ensuite vous pouvez avoir les règles métier que vous allez appliquer à ces données. La validation des données est extrêmement importante, le problème c'est qu'elle peut créer un excédent de code. Vous vous retrouvez alors facilement avec du code de validation dupliqué dans presque toute votre application. Dans cet article, nous allons découvrir un outil gratuit pour rendre notre code de validation propre, facile à créer et facile à entretenir. Avant de passer à la suite, si j'ai une chose à vous conseiller c'est : **La pratique !**

Récemment, je travaillais sur l'entité **Session** qui avait les règles ci-dessous :

- La durée d'une session est de 5 jours maximum.
- Une session ne peut pas avoir lieu un dimanche.
- La date de début d'une session ne peut pas être supérieure à la date de fin.

L'une des solutions possibles et pour laquelle j'ai opté consiste à vérifier la demande entrant dans le service d'application et à envoyer les données à l'entité uniquement si elle réussit la validation. Pour ce faire, j'ai utilisé l'outil **FluentValidation**, l'équivalent de **Joi** pour les développeurs qui travaillent sur la plateforme logicielle **Node Js**. Nous devons donc installer tout d'abord la bibliothèque **FluentValidation** ensuite toutes les règles de validation seront stockées dans des classes distinctes (dans leur constructeur respectif).

Always-valid domain model

Alors, c'est quoi le **Always-valid domain model** ?

C'est une directive qui préconise que les classes du domaine telles que les entités et les value objects restent toujours dans un état valide. Toutes les classes du domaine doivent se protéger du fait de devenir invalides.

En revanche, il existe des invariants attachés à chaque classe de domaine, donc des conditions qui doivent être vérifiées à

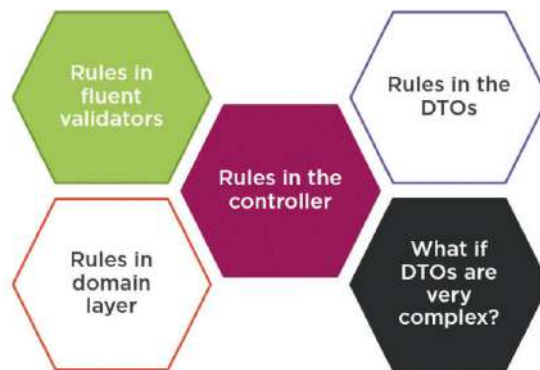


Figure 1 ["https://app.pluralsight.com/course-player?clipId=6694924b-6a48-4aaa-84ff-bcdea0bd51dc"](https://app.pluralsight.com/course-player?clipId=6694924b-6a48-4aaa-84ff-bcdea0bd51dc)

tout moment. Ces invariants définissent la classe de domaine. Par conséquent, vous ne pouvez pas violer ces invariants. Si vous le faites, la classe de domaine cessera tout simplement d'être ce que vous attendez d'elle.

Exemple 1 :

Un triangle est un concept qui a 3 arêtes. La condition **edges.Count == 3** est intrinsèquement vraie pour tous les triangles. Si vous le violez, par exemple, en ajoutant le 4ème bord, l'objet deviendrait un carré, et non pas un triangle.

Exemple 2 :

Une licorne sans corne est un cheval, pas une licorne.

Pour comprendre le raisonnement derrière cette directive, nous devons d'abord considérer leur alternative : une situation où nous autorisons les classes du domaine à entrer dans un état invalide. Pourquoi quelqu'un permettrait-il aux classes de domaine de devenir invalides ? C'est parce que c'est un moyen assez pratique de faire de la validation.

Dans notre exemple, nous validons actuellement les données entrantes telles que **SessionDto**, ce que nous pourrions faire à la place c'est de créer d'abord une **Session** en utilisant les données de la requête et ensuite valider cette **Session**. Le principal avantage de cette approche est que nous pouvons déléguer le processus de validation à la classe **Session** elle-même. Ceci est utile car la validation consiste souvent en une logique de domaine complexe. Le meilleur endroit pour placer cette logique est la couche de domaine, nos entités et nos value objects.

Un autre avantage est que cette approche nous permet de catégoriser et de regrouper facilement toutes les règles de validation dans notre projet.

Par exemple, toutes les validations liées aux notes peuvent aller à la classe du domaine **Session** et toutes les vérifications liées à une autre entité ; à la classe du domaine de cette dernière.

A retenir :

Le premier élément constitutif de la bonne technique de validation est le concept de **domaine de modèle toujours valide** (Always-valid domain model).

Toutes les classes de domaine doivent se **protéger du fait de l'invalidité**. **Figure 2**



Cylia BIBI

Apprentie en développement web chez Formiris

Passionnée par le web et les nouvelles technologies ; je me suis orientée vers le développement web. Actuellement je suis en dernière année Master 2 "Expert en développement web" à Ynov Campus et j'ai la chance de pouvoir développer mes compétences dans le cadre de mon apprentissage.

Je suis toujours prête à me lancer dans de nouvelles technologies et à découvrir celles d'actualité.

Le plus important ici, c'est notre **Domain Model**, nous devons le protéger de l'influence du monde extérieur et veiller à ne pas permettre à l'invalidité de s'infiltrer. Nous implémentons donc cette protection via la validation afin de différencier toutes les demandes. Seules les demandes valides seront autorisées à passer. Avant tout, voici une image qui résume notre processus de validation :

Tous les éléments du **Subset** sont inclus dans le **Superset**. Cette théorie des Set est utile pour parler de la validation car elle permet de visualiser un grand nombre de concepts. Nous pouvons donc définir la validation comme un processus qui consiste à essayer de déterminer si un élément dans le **Superset** a un élément correspondant dans le **Subset**.

Figure 3

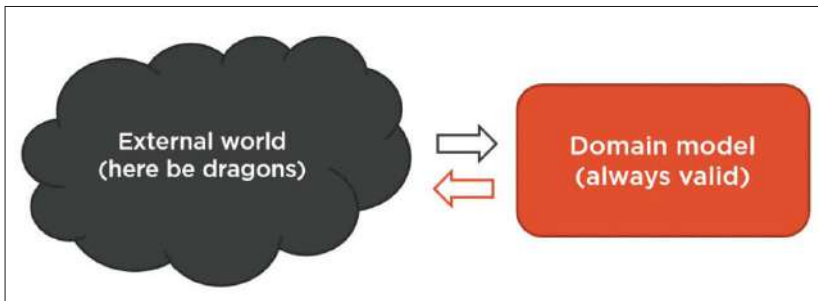
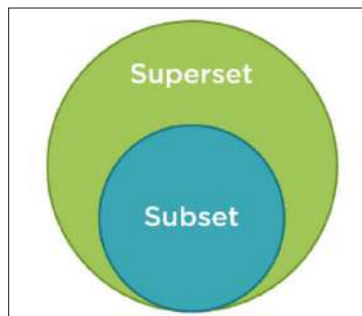


Figure 2 : Voici une image qui représente notre **Domain Model** au moment de la communication avec l'extérieur.

<https://app.pluralsight.com/course-player?clipId=6694924b-6a48-4aaa-84ff-bcdea0bd51dc>

Figure 3



<https://app.pluralsight.com/course-player?clipId=6694924b-6a48-4aaa-84ff-bcdea0bd51dc>

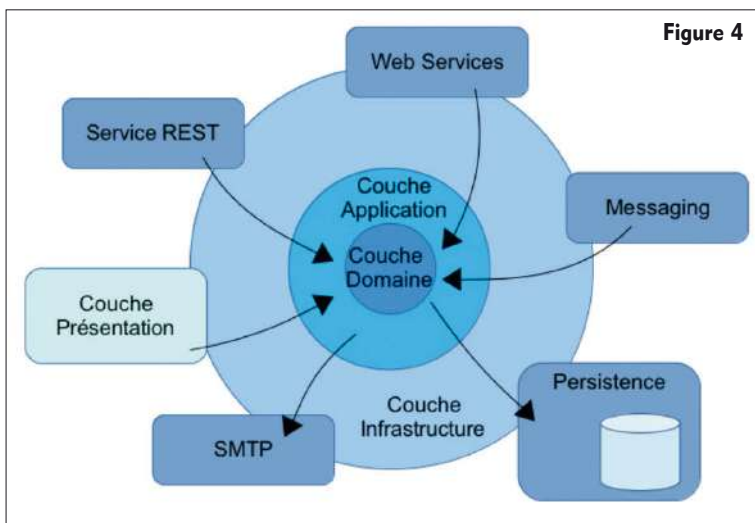


Figure 4

<https://app.pluralsight.com/course-player?clipId=6694924b-6a48-4aaa-84ff-bcdea0bd51dc>

Nous avons cité plusieurs fois la couche domaine, en quoi est-ce que ça consiste ?

La couche de modèle de domaine

La conception pilotée par le domaine (i.e. **Domain-Driven Design** ou **DDD**) est une approche de conception logicielle définie par Eric Evans qui vise à accorder de l'importance au domaine métier. En effet, dans la plupart des logiciels, la logique métier qui est implémentée constitue la plus grande valeur ajoutée puisque c'est cette logique qui rend le logiciel fonctionnel. L'approche **DDD** vise, dans un premier temps, à isoler un domaine métier. Un domaine métier riche comporte les caractéristiques suivantes :

- Il **approfondit** les règles métier spécifiques et il est en **accord** avec le modèle d'entreprise, avec la stratégie et les processus métier;
- Il doit être **isolé** des autres domaines métier et des autres couches de l'architecture de l'application;
- Le modèle doit être construit avec un **couplage faible** avec les autres couches de l'application;
- Il doit être une **couche abstraite** et assez **séparée** pour être facilement maintenue, testée et versionnée;
- Le modèle doit être conçu avec le **moins de dépendances possibles** avec une technologie ou un framework. Il devrait être constitué par des objets **POCO** (Plain Old C# Object). Les **POCO** sont des objets métier disposant de données, de logique de validation et de logiques métier. **Il ne doit pas comporter de logique de persistance**;
- Le domaine métier ne doit **pas comporter de détails d'implémentation de la persistance**.

Concevoir une architecture compatible avec le DDD

DDD préconise de séparer le code en quatre couches pour ne pas diluer la logique métier à plusieurs endroits. Chaque couche a une fonction particulière qui est utilisable par d'autres couches de façon à :

- Mutualiser le code suivant une logique;
- Éviter la duplication de code métier;

Les 4 couches sont : (Figure 4)

- la couche utilisateur (Présentation, ci-dessus),
- la couche application,
- la couche domaine,
- la couche infrastructure.

La couche domaine : [1]

Conformément aux principes **d'ignorance de la persistance** et **d'ignorance de l'infrastructure**, cette couche doit ignorer complètement les détails de la persistance des données. Ces tâches de persistance doivent être effectuées par la couche d'infrastructure. Par conséquent, cette couche ne doit pas prendre de dépendances directes sur l'infrastructure, ce qui signifie qu'une règle importante est que vos classes d'entités de modèle de domaine doivent être des **POCO**. Elle contient les informations sur le domaine et la logique métier :

- Elle détient tous les concepts du modèle métier, les cas d'utilisation et les règles métier;
- Elle contient l'état des objets métier toutefois elle n'effectue pas directement la persistance de ces objets;

- Elle peut aussi contenir l'état d'un cas d'utilisation métier si celui-ci est formé de plusieurs requêtes de l'utilisateur;
- Elle peut contenir des objets de service si leur comportement ne peut être implémenté dans un objet métier. Les services contiennent des comportements métier qui ne peuvent pas faire partie d'un objet du modèle;
- Cette couche est le cœur du métier, elle doit être isolée des autres couches et ne peut être dépendante d'un framework;
- Cette couche ne peut faire appel qu'à la couche infrastructure.

Le *Big Red Book (Vaughn Vernon, Implementing Domain-Driven Design)* explique en détail la couche de modèle de domaine et la couche d'application, il fournit ainsi un résumé rapide et aborde certaines des questions de haut niveau autour de la conception axée sur le domaine.[4]

La couche application : [1]

La couche application sépare la couche utilisateur de la couche domaine:

- Elle ne contient pas de code métier mais peut être amenée à contenir du code permettant de gérer des changements dans la couche utilisateur.
- Elle peut effectuer des validations basiques (non liées à des règles métier) sur les entrées de l'utilisateur avant de les transmettre aux autres couches de l'application.
- Elle ne doit pas contenir de logique métier ni de logique d'accès aux données.
- Cette couche peut faire appel à la couche domaine et à la couche infrastructure.

La couche infrastructure : [1]

Elle permet de fournir un lien de communication entre toutes les autres couches.

La couche utilisateur : [1]

Elle présente l'information à l'utilisateur et reçoit ses commandes. Cette couche peut faire appel à toutes les autres.

FluentValidation

Fluent Validation est une bibliothèque de validation pour .NET utilisée pour créer des règles de validation fortement typées pour les objets métier. Elle rend les validations propres et faciles à créer et faciles à lire grâce au style déclaratif adopté. Contrairement au style impératif...

Voici le code de mon entité à valider :

Code complet sur programmez.com & [github](https://github.com)

Voici la classe qui va valider les demandes de création d'une **Session** (ce qui est très attrayant, c'est la nature déclarative des règles de validation):

```
using FluentValidation;
namespace LaValidation
{
    public class SessionValidator : AbstractValidator<SessionDto>
    {
        public SessionValidator()
        {

```

```
RuleFor(s => s.Begin).NotEmpty().WithMessage("Begin date
must have a value");
RuleFor(s => s.End).NotEmpty().WithMessage("End date must
have a value");
RuleFor(s => s).MustBeValueObject(s => Session.Create(new
DateRange(s.Begin, s.End), Place.Create(s.InseeCode)));
}
}
```

Voici également le **Dto** :

```
using System;
namespace LaValidation
{
    public class SessionDto
    {
        public DateTime Begin { get; set; }
        public DateTime End { get; set; }
        public string InseeCode { get; set; }
    }
}
```

Vous remarquerez comment nous définissons différentes validations sur le même objet. C'est le style de programmation où chaque méthode retourne un objet que nous pouvons utiliser pour faire d'autres appels de méthodes. Il s'agit du **Fluent Interface pattern** qui est l'idéal pour la lisibilité du code. Quand nous créons le validateur dans notre service, ce dernier s'en charge pour mettre en place ces règles de validation centralisées dans le validator.

C'est de là que le nom de la bibliothèque **FluentValidation** provient.

Voici également une classe de tests qui fait le parallèle entre le dto et le domain model :

Code complet sur programmez.com & [github](https://github.com)

Tout cela ne veut pas dire que vous ne pouvez jamais vous protéger contre les données provenant de la base de données d'une application. Gardez à l'esprit que le concept de **garde** est différent de celui de la **validation**. Je vous encourage vivement à aller jeter un œil à l'article suivant : <https://enterprisecraftsmanship.com/posts/always-valid-domain-model/> , ainsi que le repo de **Vladimir Khorikov** où vous allez trouver Le code source du cours Pluralsight sur la validation, le DDD et la bibliothèque FluentValidation : <https://github.com/vkhorikov/ValidationInDDD>.

Conclusion

A travers cet article, nous avons abordé plusieurs éléments constitutifs de la bonne technique de validation. Notre **Domain Model** doit toujours être valide. Toutes nos règles de validation doivent appartenir à la couche de domaine. Les règles de validation dépendent du contexte.

Bibliographie

- [1] <https://cdiese.fr/domain-driven-design-en-5-min/>
- [2] <https://enterprisecraftsmanship.com/posts/always-valid-domain-model/>
- [3] <https://enterprisecraftsmanship.com/>
- [4] <http://hombredequeso.id.au/2013/02/19/implementing-domain-driven-design-1.html>



Jinane MEHDAD

Développeuse web et mobile, passionnée par les sujets autour du machine learning

RTK QUERY : un outil pour faciliter la récupération et la mise en cache de vos données

Fin 2020, une première version Alpha de RTK Query a vu le jour, les équipes de redux se sont posées la réflexion de proposer aux développeurs une façon de gérer la récupération des données sans forcément impliquer la brique globale de gestion d'état au sens "strict" de redux. L'idée étant de pouvoir récupérer facilement la donnée et de l'utiliser directement dans un composant défini, le tout dans un code se voulant concis. RTK Query utilise redux en arrière plan mais ne nécessite pas forcément une compréhension "précise" de redux pour l'utiliser. Cependant, pour ceux qui connaissent déjà redux et redux toolkit, cela offre une meilleure vision de son apport.

Les promesses de RTK Query

Lorsque nous récupérons ou que nous mettons à jour des données via une API, nous souhaitons offrir à nos utilisateurs la meilleure expérience possible. Pour ce faire, nous mettons en place différentes logiques :

- Il faut intercepter l'état de la requête "en cours", "en suspens", "terminé", etc. Notamment dans le but de faire apparaître un loader afin d'avertir l'utilisateur de l'état de celle-ci.
- Pour de meilleures performances, nous souhaitons éviter les requêtes en doublons, et cela, lorsque les données n'ont pas été altérées par une quelconque modification.
- Lors d'une mise à jour, nous voulons que l'utilisateur soit assuré que sa modification est quasi-immédiate.
- La gestion de la durée de vie du cache est aussi à prendre en considération, et dépend de la stratégie adoptée concernant la mise à jour des données.

Nous pouvons passer par l'utilisation des **thunks** de redux qui permettent d'écrire de façon asynchrone notre logique de requêtes et de dispatcher des actions en vue d'enclencher une future modification du store Redux ou nous pouvons utiliser l'une des dernières façons de faire de redux toolkit en utilisant la fonction **createAsyncThunk**.

Dans les deux cas de figure nous écrivons nous même l'ensemble des logiques mentionnées plus haut, ce qui était relativement verbeux. La promesse de RTK Query est donc de nous faciliter cela et d'intégrer en arrière plan une grande partie du mécanisme propre à ces logiques de sorte que nous n'ayons pas forcément besoin de passer par cette écriture "manuelle" pour arriver au résultat escompté.

Utiliser RTK Query dans un projet react

Ensemble, nous allons mettre en place un projet react. Il s'agira d'une application utilisant l'API "the movie database" avec pour objectif de créer des listes dans lesquelles vous pourrez ajouter des films. Nous pourrions ainsi explorer des requêtes de types GET et POST en utilisant RTK Query.

Nous allons nous faciliter la création d'un projet react en utilisant la commande create-react-app :

```
npx create-react-app mon-app
```

Auquel nous ajoutons les packages @reduxjs/toolkit, react-redux, history et react-router-dom :

```
npm install @reduxjs/toolkit react-redux history react-router-dom
```

L'API Service

Nous allons commencer par créer l'API service qui sera en charge d'ajouter une liste et de récupérer l'ensemble des listes créées, ainsi que d'ajouter un film à une liste et d'afficher l'ensemble des films dans une liste donnée. Dans un dossier src/services ajouter un fichier api.js.

A l'aide de **createApi**, nous allons définir l'url de base qui servira à l'ensemble des requêtes en utilisant notamment **fetchBaseQuery** qui enveloppe l'api fetch toujours dans une optique de simplifier la récupération des données..

Puis nous définissons trois "query endpoints" et deux "mutation endpoints".

Les queries

Les points de terminaisons (endpoints) de type query (GET) vont permettre de récupérer les données depuis un serveur. Dans notre cas :

- **getAccountList**, récupéra l'ensemble des listes existantes pour un compte utilisateur donné.
- **searchMovie**, permet de chercher des films, basé sur la recherche demandée.
- **getMoviesFromList**, affiche la liste des films rattachés à une liste.

Les Mutations

Pour mettre à jour de la donnée (PUT, PATCH, DELETE, POST) nous utilisons les mutations :

- **addList**, pour l'ajout d'une nouvelle liste liée à un compte utilisateur spécifique.
- **addMovie**, pour l'ajout d'un nouveau film dans une liste précise.

```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react'
import { BASE, API_KEY, SESSION_ID, ACCOUNT_ID } from '../app/utills'

export const apiTMDB = createApi({
  baseQuery: fetchBaseQuery({ baseUrl: BASE }),
  endpoints: (builder) => ({
    getAccountList: builder.query({
      query: () => `/account/${ACCOUNT_ID}/lists?api_key=${API_KEY}&session_id=${SESSION_ID}`,
    }),
    addList: builder.mutation({
      query: (body) => ({
        url: `/list/${id}?api_key=${API_KEY}&session_id=${SESSION_ID}`,
        method: 'POST',
        body: body
      }),
    }),
    getMoviesFromList: builder.query({
      query: (id) => `/list/${id}?api_key=${API_KEY}`,
    }),
    addMovie: builder.mutation({
      query: ({list_id,...body}) => ({
        url: `/list/${list_id}/add_item?api_key=${API_KEY}&session_id=${SESSION_ID}`,
        method: 'POST',
        body
      }),
    }),
    searchMovie: builder.query({
      query: (search) => ({
        url: `/search/movie?api_key=${API_KEY}&page=1&query=${search}&page=1`
      })
    })
  })
})
```

Notons que pour chaque endpoint un hook est automatiquement généré et pourra être utilisé dans les composants react gérant les fonctionnalités d'affichage et d'ajout de listes ou de films.

```
export const {
  useGetAccountListQuery,
  useAddListMutation,
  useGetMoviesFromListQuery,
  useAddMovieMutation,
  useSearchMovieQuery
} = apiTMDB
```

Ajouter l'api Service au store

Nous configurons notre store en utilisant la fonction `configureStore` de `redux toolkit`. Dans lequel nous ajoutons le "slice reducer" fournit par notre service API, ainsi que le middleware qui permettra de requêter l'API "the movie database"

```
import { configureStore } from '@reduxjs/toolkit';
import { apiTMDB } from '../services/list'

const store = configureStore({
  reducer: {
    [apiTMDB.reducerPath]: apiTMDB.reducer,
  },
  // adding the api middleware enables caching, invalidation, polling and other
  // features of `rtk-query`
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware().concat(apiTMDB.middleware),
});
export default store
```

Exposer le store

Il faut ensuite englober notre application à l'aide d'un `Provider` et injecter à celui-ci notre store.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import { Provider } from 'react-redux';
import { store } from '../src/app/store';

/**
 * Rendre le store disponible pour nos composants react en englobant notre composant
 * parent
 * par le composant Provider et en passant le store dans la props store
 */
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);
```

Créer un router

Nous allons naviguer sur deux principales interfaces en point d'entrée (composant `List`) nous aurons la visibilité sur les listes avec le nombre de films associés. Ainsi, qu'un bouton "+" pour créer une nouvelle liste. **Figure 1**

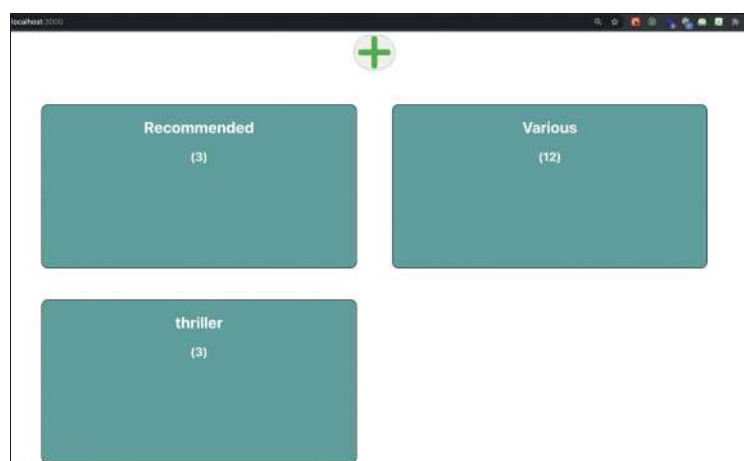


Figure 1

Et en cliquant sur l'une des listes (composant `ListDetail`) nous accédons au détail de celles-ci, contenant l'ensemble des films, dont un bouton pour ajouter un film à cette dernière.

Figure 2

Notons que pour interagir avec l'historique de notre navigateur, nous utilisons la méthode `createBrowserHistory`, en veillant à "injecter" l'objet history dans notre Router.

```
import { Switch, Route, Router as BrowseRouter } from 'react-router-dom'
import List from '../features/list/list'
import ListDetail from '../features/list/listDetail'
import { createBrowserHistory } from 'history';

const history = createBrowserHistory({});

const Router = () => {
  return (
    <BrowseRouter history={history}>
      <Switch>
        <Route exact path="/" component={List} />
        <Route path="/list/:id" component={ListDetail} />
      </Switch>
    </BrowseRouter>
  )
}

export default Router
```

Fonctionnalité autour des listes

En premier lieu, nous créons notre composant `List`, chargé d'afficher l'ensemble des listes. Nous avons vu lors de la création du service Api que des **hooks** étaient générés pour chaque endpoint. Ainsi, dans le composant `List` nous allons utiliser le hook `useGetAccountListQuery` qui va automatiquement récupérer les données au moment où le composant sera monté (mount). Nous avons alors accès au résultat via `accountList` et nous pouvons également vérifier l'état de la requête grâce aux booléens `isLoading` (requête pour la première fois, aucune donnée), `isSuccess` (la requête semble réussie et retourne des données), `isError` (détecte une erreur). Si `isError` renvoie true, l'objet `error` nous permettra d'avoir plus d'informations.

En vue d'afficher nos listes, côté code nous vérifions à l'aide du boolean `isSuccess` que la requête s'est déroulée

correctement, nous veillons aussi à ce que `accountList` contienne bien des données et nous bouclons sur le tableau de résultat renvoyé. Enfin, pour chaque liste nous récupérons l'id, dans le but d'accéder à celles-ci, et cela, au clic !

La fonction `handleAddList`, gère l'affichage de la modal `AddList` permettant de créer une nouvelle liste.

```
import React, { useState } from 'react'
import { useHistory } from 'react-router-dom';
import Button from './button/button';
import './list.css'
import AddList from './modal/addList';
import { useGetAccountListQuery } from '../services/api'

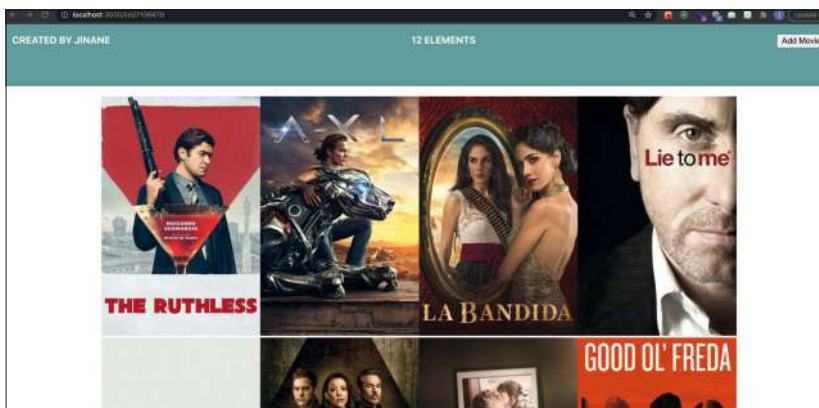
const List = () => {
  const history = useHistory()
  const [ display, setDisplay ] = useState(false)
  const { data: accountList, isLoading, isSuccess, isError, error } = useGetAccountListQuery()

  const handleAddList = () => {
    setDisplay(!display)
  }

  return (
    <div>
      {isError && <Button {...{label: 'add', handleClick: handleAddList}} />}
      <div className={isLoading ? "container-loader" : isError ? "container-error" : "container"}>
        {isLoading && '...Loading'}
        {isError && error.data.status_message}
        {isSuccess && accountList && accountList.results.length > 0 && accountList.results.map((l) =>
          <div className="container__list-card" key={l.id} onClick={() => history.push(`/list/${l.id}`)}>
            <h2>{l.name}</h2>
            <h3>{l.item_count}</h3>
          </div>
        )}
      </div>
      {display && <AddList {...{handleClose: handleAddList}} />}
    </div>
  )
}

export default List
```

Figure 2



Modal d'ajout d'une liste

Le composant propre à la modal `AddList`, fait appel au hook `useAddListMutation` qui gère donc une mutation conduisant à une création (POST) de liste. Ce **hook** donne accès à ce qu'on appelle un déclencheur (trigger) de mutation, il s'agit d'une fonction, qui une fois appelée permet de lancer la requête pour ce point d'entrée. Comme pour le précédent hook de type query, nous pouvons accéder à l'état de la requête.

En ce qui concerne la fonction `handleChange`, elle gère le **state local** en mettant à jour les valeurs renseignées dans

chacun des champs du formulaire de la modal.

C'est la fonction `handleSubmit` qui permet d'appeler le déclencheur de la mutation évoquée ci-dessus. En cas de succès s'ensuit la fermeture de la modal à l'aide du hook `useEffect`.

Code complet sur [programmez.com](#) & [github](#)

Enclencher la mutation de manière asynchrone

Il est important de remarquer qu'une fois enclenché, une promesse est renvoyée, ainsi qu'une méthode `unwrap`, qui si elle est utilisée permet d'accéder directement à la réponse avec le **payload** de succès ou l'erreur brute.

En utilisant l'asynchrone, nous pouvons décliner le code de la manière qui suit :

```
const handleSubmit = async (e) => {
  e.preventDefault()
  try {
    const payload = await addList(formVal).unwrap()
    console.log(payload)
  } catch (err) {
    console.error(err)
  }
}
```

En cas de succès la méthode `console.log()` permet d'accéder directement à la réponse suivante : **Figure 3**

Si la requête tombe en erreur nous accédons au statut et nous pourrions avoir le détail de celle-ci : **Figure 4**

Fonctionnalité autour des films

Passons maintenant au composant `ListDetail`, qui gère l'affichage des films propre à la liste consultée. Nous faisons appel au hook `useGetMoviesFromListQuery` en lui passant l'id de ladite liste pour récupérer les données, puis après quelques vérifications nous bouclons sur le tableau de résultats retournés par la requête.

Code complet sur [programmez.com](#) & [github](#)

Modal d'ajout d'un film

La modal `AddMovie` assure l'ajout d'un film en passant par une **query** et une **mutation**.

Ainsi, le hook `useSearchMovieQuery` permettra d'effectuer une recherche et de renvoyer une liste de résultats de films correspondants (nous verrons plus bas la subtilité de l'option `skip`).

Code complet sur [programmez.com](#) & [github](#)

Affichage et mise à jour des films

Le composant `SearchList` intégré dans le rendu de la modal `AddMovie` est chargé de la mise en forme de la liste de film et de déclencher au clic sur un film la fonction

`handleAddMovie` qui fait appel au déclencheur de mutation `addMovie` propre au hook `useAddMovieMutation`. Enfin, nous vérifions l'état de la requête de mutation afin de fermer la modal comme nous l'avions fait pour l'ajout d'une liste lors du succès de celle-ci.

```
import React from 'react'
import './list.css'

const SearchList = (props) => {
  const { movies, handleAddMovie } = props

  return (
    <div className="movie-searchList">
      {movies.map((e) =>
        <div className={`movie-searchList__title ${e.id}`} key={e.id} onClick={() => handleAddMovie(e.id)}>
          <h5>
            <img
              src={`https://www.themoviedb.org/t/p/w90_and_h134_bestv2/${e.poster_path}`}
              alt="logo-movie"
              width={20}
            />
            {e.original_title}
          </h5>
        </div>
      )}
    </div>
  )
}

export default SearchList
```

Zoom sur l'option skip

Notez que le hook `useSearchMovieQuery` contient un second argument optionnel `skip` qui permet de ne pas lancer la requête au "montage" du composant, cela évite d'avoir une erreur de statut 422 liée à l'absence d'une chaîne de recherche pour le paramètre `"query"`. Ainsi, la fonction `handleChange` du composant `AddMovie` passera `skip` à `false`, conduisant à exécuter le **hook** dès l'interaction de l'utilisateur avec la zone de recherche. **Figure 5**

Figure 4

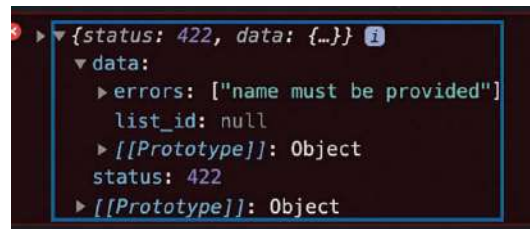
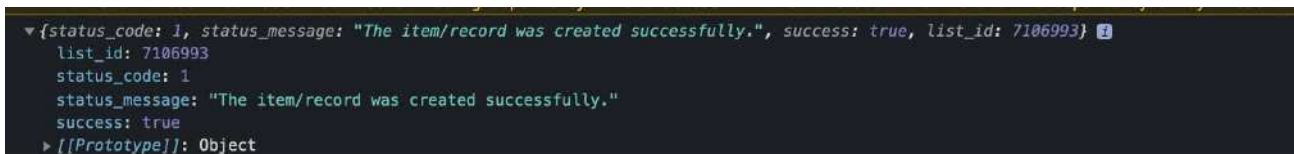
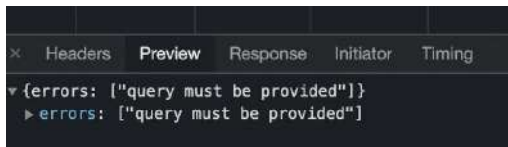


Figure 3





Invalidier le cache

Il va s'en dire que pour les lecteurs ayant déjà eu une première expérience avec RTK Query, se pose un problème dans le code de l'API Service dévoilé précédemment dans notre cheminement. En effet, si vous avez suivi pas à pas les étapes de ce projet vous pouvez constater que lors de la création d'une nouvelle liste ou l'ajout d'un film, l'interface de l'utilisateur ne se met pas à jour directement, il faut au préalable rafraîchir votre navigateur, cela est le produit du cache propre à RTK Query.

Pour faire en sorte, d'avoir une mise à jour immédiate rendant agréable l'expérience utilisateur il convient de modifier notre API service en déclarant deux **tagsTypes** "Lists" et "Movies" qui seront en charge d'invalidier le cache et de relancer automatiquement les requêtes missionnées de récupérer les listes créées et les films propres à ces listes.

Côté code, le tag de type **Lists** est fourni pour le endpoint **getAccountList**, que nous ajoutons également sous forme de tableau à la propriété **invalidatesTags** pour le endpoint mutation **addList**. Cela signifie qu'à chaque fois qu'une mutation **addList** est déclenchée, le cache relatif aux requêtes "ornées" du tag "Lists" dans la propriété **providesTags** est invalidé. Cette invalidation conduit donc à une récupération automatique de la donnée la plus à jour côté serveur.

Code complet sur programmez.com & [github](https://github.com)

Recommandation sur la gestion des API Services

Dans la documentation officielle, on nous recommande d'avoir qu'un seul API service par domaine (ou url de base) à savoir **monapi.com/api** avec plusieurs endpoints liés à cette url de base **/articles**, **/users**, etc.

Toutefois, lorsque vous avez beaucoup de points d'entrée (endpoints) pour un même domaine à gérer, il peut être pertinent d'organiser votre code en plusieurs fichiers. RTK Query vous propose de faire cela en définissant une version minimal de votre API service que vous pourrez injecter dans vos fichiers à l'aide de la méthode **injectEndpoints**. Pour plus d'informations sur celle-ci vous pouvez consulter la partie code splitting de la documentation.

Dans le cas où vous auriez plusieurs services API à gérer, il faudra rajouter dans chaque API service la propriété **reducerPath**. Par exemple, si nous avons deux URL de bases différentes pour gérer la partie liste et la partie film nous aurions fait ainsi :

```
export const MovieApi = createApi({
  reducerPath: "MovieApi",
  baseQuery: fetchBaseQuery({ baseUrl: BASE }),
```

```
export const listApi = createApi({
  reducerPath: "listApi", // We only specify this because there are many services.
  baseQuery: fetchBaseQuery({ baseUrl: BASE }),
```

Sans oublier de rajouter le slice reducer et le middleware de chaque API service dans le store redux :

```
const store = configureStore({
  reducer: {
    [listApi.reducerPath]: listApi.reducer,
    [MovieApi.reducerPath]: MovieApi.reducer
  },
  // adding the api middleware enables caching, invalidation, polling and other features
  // of `rtk-query`
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware().concat(listApi.middleware, MovieApi.middleware),
});
```

Conclusion

Nous avons vu dans ce projet comment nous pouvions utiliser RTK Query pour facilement requêter une API externe en vue de récupérer des données ou apporter des modifications sur celles-ci. RTK Query semble avoir un avenir prometteur, car il permet de réduire considérablement la logique de code et d'utiliser directement les hooks dans les composants le tout en profitant d'indicateurs sur l'état des requêtes utiles dans le rendu de l'interface utilisateur. Notons également que, la définition de l'api, ainsi que l'invalidation du cache sont centralisées au même endroit. Il devient aisé de penser de façon distinct **récupération et traitement de la donnée** avec l'usage de RTK Query et **gestion d'état** impactant plusieurs composants avec redux toolkit.

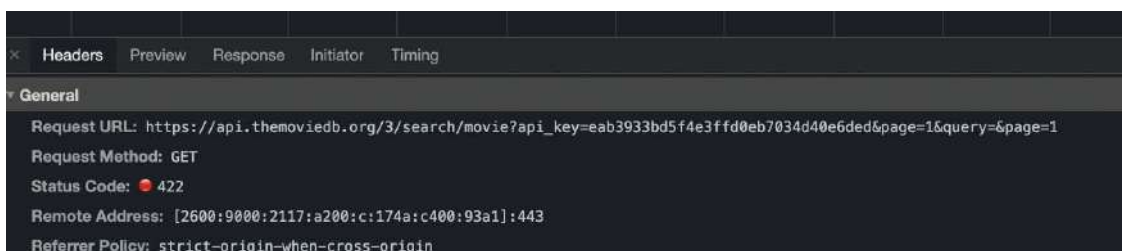
Documentation officielle de redux toolkit :

<https://redux-toolkit.js.org/rtk-query/overview>

Le projet est disponible sur le repository gitlab :

<https://gitlab.com/Mehdad2/kraken-ui-pub>

Figure 5



Ruby : 3 indispensables

Salut à toi, jeune dév. Ça y est, tu as plongé à bras-le-corps dans le monde du code, tu découvres avec joie l'élégance d'une fonction bien écrite — et avec horreur les affres de Git. Si, comme moi, tes premières années de carrière t'ont tenu bien loin des éditeurs de texte, tu ressens peut-être un frisson d'angoisse devant le fossé qu'il te reste à combler. N'aie crainte, je partage ici trois indispensables qui m'ont souvent sauvé la mise.

1 Git rebase -i, la commande qui t'épargnera des sueurs froides

J'entretiens avec Git une relation conflictuelle, faite d'amour et de rancœur, un peu comme avec ma mère. Une fois les commandes de base maîtrisées, l'outil se révèle puissant. Je passerai sur le workflow le plus classique (`git add`, `git commit`, `git push`...) pour m'intéresser à ce qui risque le plus probablement de t'arriver quand tu rejoindras une équipe de dévs : tu vas bousiller ta branche, par exemple, en perdant les modifications des autres dévs, ou en incluant un commit que tu aurais dû stasher. Ne me regarde pas comme ça. Ça arrivera. Tu le sais, je le sais, et le dév senior qui vérifiera ta branche le saura aussi... Ce qui est de loin le pire cas de figure possible, n'est-ce pas ? Fort heureusement, il existe une solution. L'utilisation de `git rebase -i`.

Installer nano

Nano est un éditeur de texte que tu ouvriras depuis ton terminal. Son installation est assez simple et dépend de ton OS. Pour ma part, j'utilise Homebrew pour l'installer, mais c'est un peu comme l'ananas sur la pizza, tu fais ce que tu veux. Tu auras peut-être besoin de mettre Nano comme éditeur Git par défaut :

```
git config --global core.editor "nano"
```

Git rebase -i

Je n'ai compris l'intérêt du rebase interactif qu'à partir du moment où j'ai commencé à utiliser régulièrement Nano (et aussi à foirer de nombreux merges). Avec la commande `git rebase -i {{hash du commit problématique}}`, l'éditeur de texte Nano démarre et affiche les différents commits. S'il y a eu un problème lors d'un précédent merge, ou si tu as rebase une mauvaise branche, tu peux donc supprimer les commits fautifs en écrivant simplement `d` devant le nom du commit. Ensuite, tu peux sauvegarder avec la commande `ctrl+O`. Pour quitter Nano, ce sera la commande `ctrl+X` qui te renverra sur le bon vieux terminal. Je conseille très fortement de s'entraîner avec Nano et le rebase interactif avant de tenter des acrobaties en production. L'excellente interface Learn Git Branching, qui permet de pratiquer Git en jouant, est un bon point de départ.

2 Postman, ton nouveau meilleur ami

Quand j'ai démarré ma reconversion vers le métier de dév, il y a un an, je me rêvais développeuse front. Je pensais donc pouvoir faire l'impasse sur l'apprentissage des requêtes aux API et toutes ces sortes de choses. Que nenni ! Même le Product Owner doit comprendre comment fonctionne une requête HTTP. Et bien souvent, pour déboguer une application, on passe

par l'éreintante vérification des erreurs renvoyées par l'API, les 404, 309, et autres joies... C'est là qu'intervient Postman.

Postman est un logiciel de requêtage d'API. Il est gratuit, fonctionnel, et diablement efficace. Il demande un peu de temps de prise en main, car il offre de nombreuses fonctionnalités et l'interface peut être un peu déroutante. Je recommande d'avoir bien en tête la structure d'une requête avant de se lancer dans l'utilisation de Postman. On peut par exemple grouper les requêtes dans des dossiers (appelés "Collections"), importer des environnements (staging, production, etc.), créer des tests...

Pour débiter avec Postman, tu peux créer une petite API dans le langage de ton choix, et la tester avec Postman. Quand j'ai débuté avec TypeScript, j'ai suivi un tutoriel Udemy qui utilisait Postman pour tester l'API qu'on construisait pendant la formation — parfait pour commencer en douceur. Postman propose aussi des formations gratuites.

3 Mon environnement préféré

Tout.e.s les développeur.euse.s que j'ai rencontrés pendant ma carrière ont essayé de me convaincre d'adopter leurs outils. Parfois, j'ai obéi et découvert des logiciels, raccourcis, commandes que j'utilise encore aujourd'hui. Parfois, j'ai perdu un temps infini à configurer des éditeurs de textes obscurs, écrire des raccourcis personnalisés pour les oublier aussitôt, ou demander pour la quinzième fois à un CTO exaspéré de refaire un process qu'il m'avait montré la veille. L'informatique adore les querelles de chapelles. Je ne me lancerai pas dans le débat espaces VS tabs, car je ne suis pas complètement maso. Je décrirai simplement ici mes outils préférés en tant que développeuse junior - le plus important, c'est d'explorer les possibilités pour découvrir les tiens !

- **iTerm2**, un très bon terminal pour macOS
- **Homebrew**, un gestionnaire de paquets pour macOS, entièrement gratuit, d'une utilisation simplissime (on installe des programmes entiers avec une seule ligne de commande !)
- **Atom**, un IDE entièrement open source, mais je ne crache pas non plus sur...
- **Visual Studio Code**, un IDE puissant et fonctionnel, open source lui aussi et bourré de plugins fort utiles
- **Zsh**, un shell plus facile à prendre en main que bash, il dispose notamment de fonctionnalités d'autocomplétion
- **Oh-my-zsh**, un sympathique framework pour les utilisateurs de zsh
- **Nano**, tellement plus simple d'utilisation que Vim (utilisateurs de VIM, RDV à 10h sur le parking du Auchan)

Pour finir...

L'informatique est un univers merveilleux, mais il peut être impitoyable. N'hésite pas, jeune dév, à demander de l'aide autour de toi, et à te créer un bon réseau.



Lucille James

Développeuse, avec un sombre passé de Product Owner. Elle aime le Ruby, les commits propres, et noyer ses profiteroles sous deux litres de chocolat.



Célia Doolaege

Je suis développeuse fullstack depuis 7 ans, chez SFEIR depuis 3 ans. Je suis un peu touche-à-tout, fortement intéressée par la qualité du code, l'approche agile et le travail en équipe. Je m'intéresse également au numérique responsable et l'écoconception, qui visent à réduire l'impact environnemental du numérique.

NestJS: un framework Node.js complet ?

De nos jours, l'informatique est un milieu qui évolue si rapidement que le time-to-market (la durée entre la conception d'une idée et le moment où le produit fini arrive entre les mains des utilisateurs) est devenu un indicateur primordial. Pour produire rapidement des fonctionnalités, les développeurs se basent depuis longtemps sur des frameworks, qui sont conçus pour augmenter la productivité en prenant en charge une partie de la complexité du développement.



NestJS (Nest) est un framework open source conçu pour développer des applications sur la plateforme Node.js. Il est écrit en Typescript qu'il supporte nativement, bien qu'il permette aussi de développer son application en JavaScript. Le véritable avantage de Nest est d'accélérer le démarrage du projet, en proposant une architecture inspirée d'Angular qui permet aux équipes de développer des applications facilement testables, scalables et maintenables dans le temps. Il compte presque un million de téléchargements hebdomadaires sur npm en novembre 2021. Son fonctionnement peut être comparé à celui de Spring pour Java, avec un système d'annotations et d'injection de dépendances. Nest a une [documentation](https://docs.nestjs.com/cnaire) (<https://docs.nestjs.com/cnaire>), fournie et détaillée.

Dans cet article, nous allons voir ensemble un exemple d'application écrite avec Nest : la gestion d'une liste personnelle de séries avec notes et commentaires. Cette application permet de lister les avis sur des séries et de créer un nouvel avis à l'aide d'un formulaire.

Codes de cet article sur <https://github.com/CeliaDoolaege/my-list-of-series>

Premiers pas et configuration

En tant que framework, Nest a fait des choix en amont pour que les développeurs n'aient pas à faire la configuration du projet eux-mêmes, ce qui est une étape souvent longue à mettre en place et assez pénible, mais qui n'apporte aucune valeur métier. Nest fournit donc une [CLI](https://docs.nestjs.com/first-steps) (<https://docs.nestjs.com/first-steps>) qui va créer rapidement et facilement une application de base, déjà configurée et prête à l'emploi, avec l'arborescence suivante :

Figure 1.

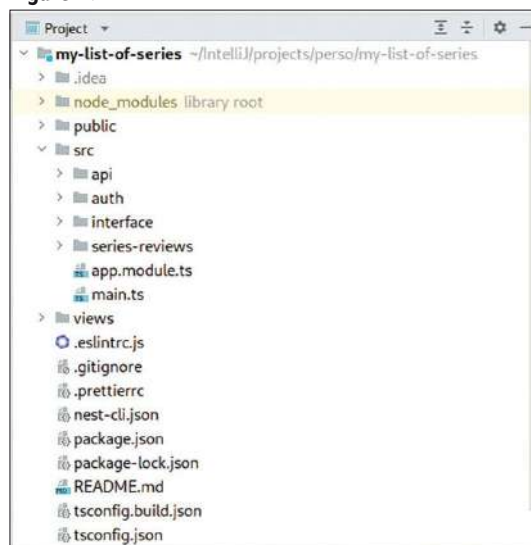


Figure 1

Le projet généré marche immédiatement, il suffit de le lancer avec `npm start`, et nous avons déjà une application qui tourne sur localhost:3000, même si elle ne fait qu'afficher "Hello World" dans le navigateur.

Nest fournit nativement la configuration de Typescript, ESLint et Prettier, qui s'occupent respectivement du typage de JavaScript, de la vérification des conventions de code et du formatage. Ces configurations restent modifiables si besoin, et même supprimables comme n'importe quelle autre dépendance. Ce sont des outils qui sont largement utilisés par la communauté des développeurs JavaScript car ils facilitent la gestion du projet et surtout sa maintenabilité dans le temps.

Dans le package.json, un certain nombre de scripts sont déjà définis, notamment les scripts nécessaires pour démarrer l'application (avec hot reload pour la phase de développement), faire tourner ESLint et Prettier, ou encore lancer les tests. Nest installe et configure par défaut le framework de test [Jest](https://github.com/facebook/jest) (<https://github.com/facebook/jest>), le plus répandu sur les applications JavaScript. Si on lance le script `npm test`, nous avons déjà 1 test qui passe, qui est là pour l'exemple. Des tests end-to-end sont également présents dans le dossier test. Nous pouvons bien sûr installer en plus n'importe quelle dépendance souhaitée, comme dans n'importe quel projet Node.js.

Performances

Nest est construit au-dessus d'[Express](https://expressjs.com) (<https://expressjs.com>), le framework Node.js open source le plus répandu. Mais si la performance est au cœur de vos préoccupations, Nest est également compatible avec [Fastify](https://www.fastify.io) (<https://www.fastify.io>), un autre framework open source centré sur la [performance](https://docs.nestjs.com/techniques/performance) (<https://docs.nestjs.com/techniques/performance>).

Les modules

La première complexité dans un projet, c'est l'architecture : pour assurer la maintenabilité du projet dans le temps, il faut une structure claire et scalable. Il faut limiter au maximum l'entropie, c'est-à-dire la tendance naturelle des projets informatiques à se complexifier avec le temps, avec un impact sur la productivité dans le développement de nouvelles fonctionnalités.

Nest a fait le choix d'une architecture modulaire : chaque fonctionnalité sera vue comme un [module](https://docs.nestjs.com/modules) (<https://docs.nestjs.com/modules>). Un module se compose d'abord d'un ou plusieurs controllers, qui exposent des routes. Un module contient des providers, qui sont des classes à comportement

(métier, base de données, etc.). Un module peut exporter des classes et être importé dans d'autres modules. Chaque module contient tout ce qui est nécessaire à son fonctionnement.

Prenons par exemple une fonctionnalité qui servirait juste à créer un avis sur une série. Nous créons un module `CreateReviewModule` qui expose une route permettant de noter une série en laissant un commentaire :

```
@Module({
  controllers: [CreateReviewController],
  imports: [
    MongooseModule.forFeature([
      { name: SeriesReview.name, schema: SeriesReviewSchema },
    ]),
  ],
  providers: [CreateReviewRepository, CommentChecker],
})
export class CreateReviewModule {}
```

Ici, on voit que notre module expose un controller `CreateReviewController` qui contient la route. Il importe le module `Mongoose` (<https://mongoosejs.com>), un ORM qui gère pour nous le mapping entre nos entités et la base de données MongoDB dans laquelle nous allons stocker les notes et commentaires des séries. Enfin, nous voyons dans les providers deux classes `CreateReviewRepository`, qui sont chargées de la sauvegarde en base de données, et `CommentChecker`, qui sera chargé de vérifier que le contenu du commentaire est autorisé (par exemple, pour éviter de sauvegarder un commentaire avec du langage injurieux).

Toutes les classes qui sont répertoriées dans les providers peuvent ensuite être injectées dans les controllers ou les autres providers. Les classes exportées par les modules que nous importons peuvent également être injectées dans les classes de notre module. Dans cet exemple, on voit facilement le périmètre de notre fonctionnalité : toutes les dépendances de notre controller sont listées dans ce module. Lorsqu'on parle de maintenabilité dans le temps, la capacité à anticiper les impacts de changements dans notre code compte beaucoup, et l'architecture préconisée par Nest permet de plus facilement prédire les impacts de nos changements.

Cette architecture est également scalable, car l'ajout de nouveaux modules n'impactent pas ceux qui sont déjà présents, chaque nouvelle fonctionnalité vient juste s'ajouter dans le root module, c'est-à-dire celui qui va ensuite importer tous les autres modules. La complexité locale dans les modules reste liée à la complexité métier, et non à la taille du projet.

Par exemple, dans notre projet, nous pouvons imaginer deux modules : un pour lister les avis existants et un autre pour créer un nouvel avis. Les deux modules se servent du même module `Mongoose` pour la base de données, mais peuvent aussi avoir besoin d'autres modules propres, par exemple pour récupérer les affiches des séries dans la liste des avis. Chaque module n'importe que ce dont il a besoin dans un souci de responsabilité limitée. **Figure 2**

Injection de dépendances

Avant d'aller plus loin, faisons un petit aparté sur l'injection de dépendances. À la base, c'est le cinquième des principes

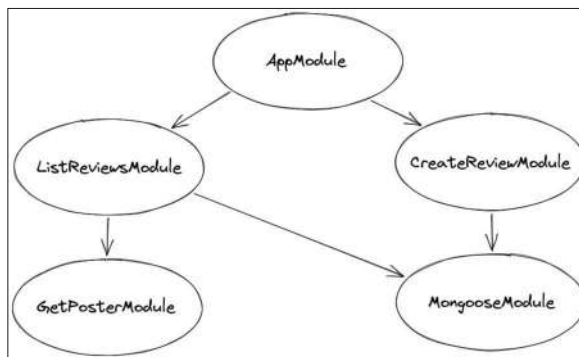


Figure 2

SOLID de la Programmation Orientée Objet (D pour Dependency Inversion). L'idée est qu'une classe de "haut niveau" (gestion des règles métier) ne soit pas directement liée à une classe de "bas niveau" (gestion de l'infrastructure). Par exemple, on crée une interface avec des fonctions de lecture en base de données, et on injecte dans les classes métier une classe qui implémente cette interface.

Ce qui est intéressant ici, c'est que notre classe métier n'a pas la responsabilité d'instancier la classe qui lit en base de données, elle s'attend à avoir une classe qui respecte la bonne interface et peut donc appeler ses fonctions sans se soucier de l'implémentation. Notre classe métier n'a pas besoin de savoir que cette implémentation est en MongoDB ou PostgreSQL, ou encore un mock pour les tests unitaires (nous y reviendrons dans le paragraphe sur les tests). On sépare bien les responsabilités de chaque classe.

En tout cas, c'est sur ce principe que se base Nest : en déclarant une classe dans un module, elle devient injectable dans les autres classes du module. Maintenant, nous allons voir concrètement comment construire le code autour de ce principe.

Controller et validation

Créons maintenant une route pour donner son avis sur une série. Il s'agit d'une route POST puisque nous créons un nouvel avis. Un avis contient le titre de la série, une note comprise entre 0 et 5 et un commentaire optionnel.

La première chose à faire (en dehors des tests si vous faites du TDD, ici nous y reviendrons après) est de créer la route d'ajout de commentaire. C'est le rôle du [controller](https://docs.nestjs.com/controllers) (<https://docs.nestjs.com/controllers>) qui va répondre lors d'un appel à la route. Nest fournit les annotations nécessaires pour créer une route Post, récupérer le body et retourne automatiquement un statut 201 Created si aucune exception n'est renvoyée.

Il ne reste donc au développeur qu'à implémenter le vrai code métier, à savoir vérifier que si un commentaire est présent, alors il doit être valide (sans contenu injurieux), puis sauvegarder cet avis en base de données.

```
@Controller()
export class CreateReviewController {
  constructor(
    private commentChecker: CommentChecker,
    private createReviewRepository: CreateReviewRepository,
  ) {}

  @Post('/series/reviews')
```



```

async grade(@Body() gradeRequest: ReviewRequest): Promise<void> {
  if (gradeRequest.comment) {
    const isValidComment = this.commentChecker.check(gradeRequest.comment);

    if (!isValidComment) {
      throw new BadRequestException({
        message: 'This comment is not acceptable',
      });
    }
  }

  await this.createReviewRepository.save(gradeRequest);
}

```

Comme on peut le voir ici, les classes `CommentChecker` et `CreateReviewRepository` sont des dépendances injectées par le constructeur, qui est géré par Nest grâce au module que nous avons déclaré plus tôt.

L'annotation `@Post()` est suffisante pour déclarer la route à Nest. L'annotation `@Body()` permet de récupérer le body qui est envoyé dans le Post, qu'on peut directement typer. On renvoie `Promise<void>`, car Nest s'occupe de renvoyer un statut 201 par défaut sur les routes Post, bien qu'on puisse surcharger ce comportement si besoin.

Finalement, en plus des annotations, nous n'avons écrit que les règles métier de gestion des avis, et c'est ce qui compte : passer du temps sur la valeur métier apportée par notre code, et non la forme pour le faire fonctionner, qui est gérée par le framework. Il ne reste qu'à implémenter les fonctions dans les classes `CommentChecker` et `CreateReviewRepository` et nous avons là une route opérationnelle.

À noter ici que si le commentaire est invalide, nous renvoyons une exception de type `BadRequestException`, qui contient le statut 400 Bad Request et dans lequel nous passons juste un message explicatif.

Validation du body

Quand on soumet une requête, il faut d'abord valider que le body soumis répond à nos spécifications : tous les champs obligatoires doivent être présents, la note doit être numérique, etc. Il existe deux dépendances "class-validator" et "class-transformer" qui permettent d'assurer cette validation (<https://docs.nestjs.com/techniques/validation>) à travers des annotations sur la classe du body. Ici, nous appliquons des règles de validation sur la classe `ReviewRequest` :

```

export class ReviewRequest {
  @ApiProperty({ description: 'Title of the series' })
  @IsNotEmpty()
  title: string;

  @ApiProperty({ description: 'Grade between 0 and 5' })
  @IsNumber()
  @Min(0)
  @Max(5)
  grade: number;

  @ApiPropertyOptional({ description: 'A comment on the series' })

```

```

@IsOptional()
@IsNotEmpty()
comment?: string;

constructor(title: string, grade: number, comment?: string) {
  this.title = title;
  this.grade = grade;
  this.comment = comment;
}

```

Chaque champ se voit associer ses règles de validation. Le titre ne doit pas être vide. La note doit être numérique et sa valeur doit être comprise entre 0 et 5. Le commentaire est optionnel, mais s'il est présent, il ne doit pas être vide. Les annotations sont très explicites ici, et permettent de mettre en place les règles de validation les plus simples.

Si la validation du body échoue, alors Nest renvoie un statut 400 Bad Request avec un message qui indique quel champ est en erreur et pour quelle raison.

Il est aussi possible de faire des validations sur des tableaux, vérifier qu'il n'est pas vide, que chaque élément du tableau correspond aux règles énoncées, etc. Les annotations disponibles sont très riches.

Et si ma validation est plus complexe ?

Parfois, nous avons besoin d'exprimer des règles qui ne font pas partie des annotations de validation proposées par défaut. Dans ce cas, il est d'abord possible de créer ses propres annotations pour exécuter une vérification spécifique sur un champ. Par exemple, on peut vérifier qu'une chaîne de caractères commence bien par un mot spécifique si c'est notre besoin.

Mais on peut aussi imaginer une validation qui nécessiterait de lire plusieurs champs. Par exemple, dans notre cas, si la note associée à un avis est basse, on peut exiger que le commentaire soit obligatoire pour justifier cette note, tout en le laissant optionnel sinon. Comment gérer ce cas ?

On peut créer un [Pipe](https://docs.nestjs.com/pipes) (<https://docs.nestjs.com/pipes>) de validation. C'est une classe dont le comportement s'exécute avant que le contrôleur ne récupère le body. Elle a accès à l'objet en entrée en entier et laisse le soin au développeur d'écrire les règles de validation. Nous pouvons donc implémenter de cette manière n'importe quelle règle de validation d'un objet pour s'assurer qu'il soit bien valide lorsqu'il arrive dans le contrôleur. Dans notre exemple, si la note est inférieure à 3 et qu'il n'y a pas de commentaire, alors nous renvoyons une `BadRequestException`, sinon l'objet est valide.

```

@Injectable()
export class MandatoryCommentOnBadGradePipe implements PipeTransform {
  transform(value: unknown): ReviewRequest {
    const reviewRequest = plainToClass(ReviewRequest, value);

    if (reviewRequest.grade < 3 && !reviewRequest.comment) {
      throw new BadRequestException(
        'Comment is mandatory when grade is lower than 3',
      );
    }
  }
}

```

```

return reviewRequest;
}
}

```

Swagger

Les plus attentifs l'auront remarqué : à quoi servent les annotations `@ApiProperty()` ?

Une fois que notre route est en place, nous avons envie de la tester. Bien sûr, nous pouvons utiliser curl, Postman ou n'importe quel autre outil permettant de faire des appels d'API. Mais l'écosystème autour de Nest fournit des dépendances permettant de générer dynamiquement la documentation [swagger](https://docs.nestjs.com/openapi/introduction#bootstrap) (<https://docs.nestjs.com/openapi/introduction#bootstrap>) à partir des annotations.

La mise en place est très simple, il suffit de quelques lignes dans le fichier `main.ts` pour que cette documentation soit déployée sur une route de notre application.

Pour notre route de création d'un avis, le rendu donnerait : **figure 3**

Le schéma du body est directement généré par les annotations `@ApiProperty()` et `@ApiPropertyOptional()` et la description qu'ils contiennent. On obtient une documentation standard, facile à partager, car directement hébergée sur notre application, et facile à utiliser grâce à l'option "Try it out" (nous reviendrons sur l'authentification par la suite).

Tests unitaires

Chose promise chose due, nous allons parler maintenant des tests unitaires. Pour qu'une application reste maintenable dans le temps, il ne suffit pas que l'architecture nous aide à comprendre les fonctionnalités impactées par nos changements, il faut aussi que des [tests](https://docs.nestjs.com/fundamentals/testing) (<https://docs.nestjs.com/fundamentals/testing>) (unitaires et/ou end-to-end) soient présents pour assurer que nos changements ne créent pas de régressions dans les règles métier déjà existantes.

Grâce à l'injection de dépendances évoquée plus tôt, les classes implémentées sont facilement testables unitairement, car les dépendances peuvent être mockées, c'est-à-dire remplacées par des fausses instances où nous contrôlons le comportement et les retours.

Pour tester un controller, Nest fournit les outils pour créer des modules de test, où l'on peut injecter nos dépendances mockées :

Code complet sur [programmez.com](#) & [github](#)

Ici, nous créons une fausse instance de `CommentChecker` et de `CreateReviewRepository`, nous utilisons Jest pour la fausse implémentation des fonctions de ces deux classes, et nous les fournissons en providers au module de test. Ensuite, il ne reste dans le test qu'à appeler la route et vérifier le retour.

Nous pouvons ensuite créer des tests pour tous les cas gérés par notre code: renvoyer une erreur si l'un des champs obligatoires est manquant, si la note n'est pas comprise entre 0 et 5, si le commentaire est injurieux, etc.

Bien sûr, les tests peuvent parfaitement être écrits avant l'implémentation, comme préconisée par le TDD (Test Driven Development).

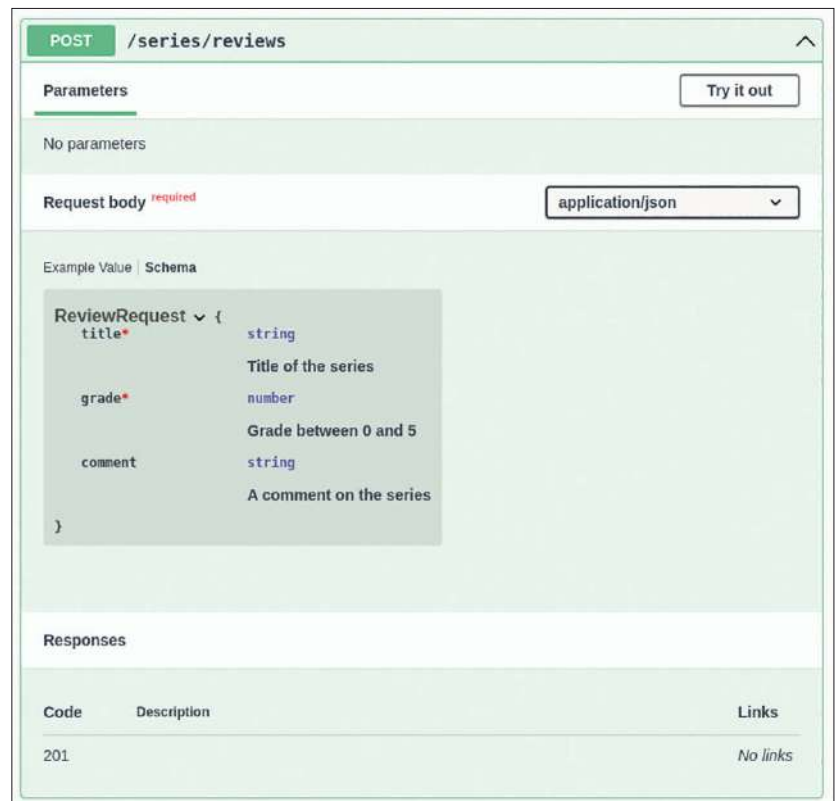


Figure 3

Sécurité et Authentification

La plupart des applications ne sont pas accessibles librement au grand public et nécessitent donc d'être sécurisées. Les préconisations classiques, telles que l'installation de la dépendance [helmet](https://www.npmjs.com/package/helmet) (<https://www.npmjs.com/package/helmet>) pour préconfigurer les headers HTTP par exemple, sont toujours de mise et ne doivent pas être oubliées. Elle fait d'ailleurs partie des [recommandations de sécurité](https://docs.nestjs.com/security/helmet) (<https://docs.nestjs.com/security/helmet>) de Nest.

Pour gérer l'authentification, dans une application Node.js en express par exemple, nous pourrions utiliser un middleware spécifique, c'est-à-dire une fonction qu'on applique sur des routes et qui s'exécutent avant que les controllers ne soient appelés. Dans Nest, les [middlewares](https://docs.nestjs.com/middleware) (<https://docs.nestjs.com/middleware>) existent aussi, ils ont la même définition, mais ne sont pas la solution idéale préconisée.

Les [guards](https://docs.nestjs.com/guards) (<https://docs.nestjs.com/guards>) marchent sur le même modèle, mais ont l'avantage de connaître le contexte dans lequel ils sont appelés : ils savent quelle route est appelée, mais aussi quel controller sera exécuté si la validation passe. Un guard peut se faire injecter une dépendance, par exemple un service qui gère la vérification d'un token.

Ici, nous avons un exemple de guard qui protège les routes à l'aide d'une authentification de type Basic, c'est-à-dire que les requêtes HTTP ont un header `authorization` qui contient `username` et `password` encodés en base 64. On vérifie ensuite que l'utilisateur est bien reconnu par l'application :

Code complet sur [programmez.com](#) & [github](#)

L'authentification Basic n'est pas la méthode la plus sûre, mais ce modèle est compatible avec d'autres techniques d'[authentification](https://docs.nestjs.com/security/authentication) (<https://docs.nestjs.com/security/authentication>) comme JWT.

Pour appliquer ce guard, il suffit d'ajouter à nos contrôleurs l'annotation `@UseGuard(AuthGuard)`. Nous aurions aussi pu définir ce guard en global dans le module `AppModule`. Nos routes sont désormais sécurisées, et le `SwaggerModule` peut prendre une option qui permet de saisir l'authentification basique directement depuis le swagger.

Interface avec Nest MVC

Nous avons maintenant une route pour donner un avis sur une série, mais le swagger n'est pas vraiment adapté pour la plupart des utilisateurs non développeurs... L'idéal serait de créer un petit formulaire qui soumet l'avis à notre API.

On peut bien sûr brancher une interface externe à nos APIs. Nest est compatible avec toutes les dépendances npm, comme `cors` (<https://www.npmjs.com/package/cors>) par exemple, qui autorise les appels cross-origin entre un frontend et un backend qui ne seraient pas hébergés sur le même domaine. Sinon, Nest permet d'implémenter toutes les facettes du MVC (Model-View-Controller) : nous avons déjà vu les parties Model et Controller précédemment, mais nous pouvons aussi implémenter la partie View directement. Il s'agit ici de faire des vues simples avec un langage de templating (type `handlebars` ou `ejs`) pour faire du SSR (Server-Side Rendering). Pour des interfaces complexes ou hautement dynamiques, cela ne sera peut-être pas suffisant, mais pour notre formulaire, ce sera parfait.

Tout d'abord, il faut écrire le fichier `handlebars` qui contiendra notre formulaire. Il s'agit là d'une page de html classique avec du templating de type `mustache`, dans lequel on peut ajouter du `css` pour le design et du `js` pour les comportements, par exemple pour vérifier les valeurs des champs obligatoires avant la soumission du formulaire.

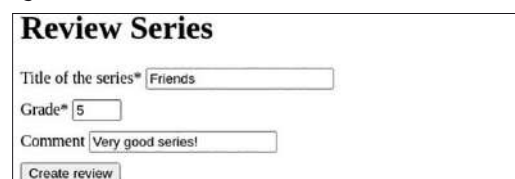
Du point de vue de Nest, notre interface est un module comme les autres, qu'il faut donc importer dans `AppModule`. Notre contrôleur fait simplement le lien entre le fichier `create-review.hbs` et la route `"/interface"` dans le navigateur :

```
@Controller()
export class CreateReviewFormController {
  @Get('/interface')
  @ApiExcludeEndpoint()
  @Render('create-review')
  createReviewForm(): void {
    // Rendering form
  }
}
```

Si nous avons besoin d'injecter des valeurs dans la page à l'aide du templating, il suffit que le contrôleur renvoie un objet contenant les valeurs à afficher. Ici, nous n'en avons pas besoin. L'annotation `@ApiExcludeEndpoint` évitera à cette route propre à l'interface de se retrouver dans le swagger.

Lorsque nous entrons l'url <http://localhost:3000/interface> dans le navigateur, nous pouvons désormais voir notre formulaire :

figure 4



Review Series

Title of the series*

Grade*

Comment

Le design est très simple dans cet exemple, mais l'important est d'avoir une interface qui permet à des utilisateurs ne maîtrisant pas swagger d'utiliser notre application. On peut bien sûr faire des interfaces bien plus jolies que celle-ci !

On pourrait d'ailleurs l'étendre avec une autre page permettant de lister les séries pour lesquelles un avis a été posté, afficher dans un encart la série ayant obtenu la meilleure moyenne, etc. Chaque écran supplémentaire sera simplement un module de plus à ajouter dans l'application.

Les points forts et points faibles de NestJS

NestJS possède de nombreux avantages lorsqu'il s'agit de démarrer une nouvelle application. D'abord, la CLI permet d'avoir immédiatement un projet opérationnel. L'architecture modulaire préconisée permet l'évolutivité et la maintenabilité dans le temps, en gardant la maîtrise de la complexité. Nest permet l'utilisation de n'importe quelle dépendance externe et ne se ferme pas à de nouveaux usages. La communauté est très réactive et de nombreux cas d'usage sont documentés.

En revanche, le framework est très riche et complexe, et on peut facilement se perdre dans la documentation lorsqu'on bloque sur un point très précis. D'ailleurs, il n'est pas rare de devoir rechercher sur Google comment faire une chose précise (par exemple, injecter un service dans un guard) plutôt que de se baser sur la documentation. D'ailleurs, cette documentation manque parfois de conseils sur des bonnes pratiques pour justement garantir la maintenabilité du projet.

Pour aller plus loin

NestJS propose encore beaucoup d'extensions qui permettent d'enrichir son projet et que je n'ai pas présentées ici, mais qu'il peut être intéressant de découvrir. On peut citer par exemple des préconisations pour la mise en place de `CORS` (<https://docs.nestjs.com/recipes/cors>) ou de `health checks` (<https://docs.nestjs.com/recipes/terminus>), ou encore l'outil de génération de documentation `Compodoc` (<https://docs.nestjs.com/recipes/documentation>).

Conclusion

Nest est un framework sur lequel je travaille personnellement au quotidien et qui tourne en production d'un site e-commerce connu. Il facilite grandement mon travail de développeuse, car il apporte des réponses prêtes à l'emploi à des questions que tout projet soulève à un moment donné : évolutivité et maintenabilité dans le temps, sécurité, authentification, etc. Le framework est très riche, et ce qu'il ne fait pas peut être géré par des outils externes, car il n'est pas fermé à l'extension à travers d'autres dépendances.

Il fait ce qu'on attend principalement d'un framework, c'est-à-dire qu'il nous soulage de la complexité de l'architecture du projet et nous laisse nous concentrer sur la complexité métier de notre application, celle qui apporte de la valeur à nos utilisateurs.

Il serait peut-être temps de changer de mot de passe...

2021 a connu la plus grande fuite de mots de passe jamais enregistrée avec la publication de *RockYou2021*, une liste contenant plus de 8 milliards de mots de passe accumulés au fil des attaques de ces dernières décennies (LinkedIn, Facebook, Sony, entre autres). L'occasion de repenser sa stratégie de choix de mots de passe, de (re)découvrir comment ils sont stockés, et en quoi une telle liste peut aider à les *cracker*.

Les mots de passe sont de plus en plus boudés par les experts en cybersécurité. Aucun n'est infaillible, tous peuvent théoriquement être *crackés* : ce n'est qu'une question de patience et de puissance de calcul. Pour renforcer leur sécurité, les applications des géants du Web démocratisent l'authentification multifactorielle (code temporaire, biométrie, clé physique, etc.). Et bien que la tendance soit à l'adoption du *passwordless*, les mots de passe demeurent ancrés dans notre quotidien, et pas si près d'en disparaître.

Se pose donc la question de la sûreté de leur stockage. Il y a quelques années à peine, certains grands services conservaient encore les mots de passe "en clair" dans leur base de données ! Un simple accès au serveur suffisait à exposer les identifiants de tous les utilisateurs. Chiffrer les mots de passe n'est pas idéal non plus, car cela ne fait que déplacer le problème : où et comment stocker la clé de chiffrement de façon sûre ? Vous pouvez avoir recours à l'algorithme le plus complexe et posséder le mot de passe le plus robuste, si la clé tombe entre de mauvaises mains, la partie est perdue. La solution retenue : utiliser les fonctions de hachage.

Un peu de cuisine théorique : hachez, salez, poivrez !

Derrière ce terme un peu obscur se cachent des noms que vous connaissez peut-être : MD5, SHA-1, en sont des exemples.

Une fonction de hachage regroupe un ensemble d'opérations qui transforment une donnée d'entrée en une empreinte numérique caractéristique, le *hash*. Quand un utilisateur crée un compte sur un service, c'est le *hash* de son mot de passe qui est stocké en base de données. Lors d'une future authentification, le serveur calcule le *hash* du mot de passe entré par l'utilisateur et le compare à celui stocké. S'ils sont identiques, le serveur valide l'utilisateur et lui donne accès à son compte.

La fonction de hachage doit donc avant tout être *déterministe* : une même entrée produira systématiquement la même empreinte en sortie. Mais sa qualité repose également sur d'autres critères :

Être irréversible

Les fonctions de hachage sont à sens unique : une fois hachée, la donnée d'entrée ne doit pas pouvoir être retrouvée en faisant le chemin inverse (dans un temps *raisonnable*, car il serait théoriquement possible de tester tous les chemins inverses). Cela peut passer par des

opérations irréversibles (par exemple, on ne peut pas retrouver de source sûre x à partir de x^2 , car x et $-x$ conduisent à ce même résultat).

Comporter un nombre élevé d'opérations

Qu'il soit pratiqué avec plus ou moins de finesse, le *cracking* de mot de passe repose toujours sur de la puissance de calcul. Les ordinateurs modernes sont de plus en plus performants, et peuvent réaliser plusieurs millions d'opérations par seconde. Utiliser une fonction de hachage complexe permet de diminuer le nombre de mots de passe testés par seconde.

Minimiser le nombre de collisions

Beaucoup de fonctions de hachage proposent un résultat de longueur fixe, peu importe l'entrée (MD5, par exemple, renvoie systématiquement une chaîne de 32 caractères, SHA-1 une chaîne de 40). Afin de garantir au maximum une relation d'unicité entre entrée et sortie, il est indispensable de limiter le nombre de collisions, autrement dit, éviter que deux entrées différentes produisent une même empreinte en sortie.

Décorrélér les modifications de l'entrée et de la sortie

Une infime modification de l'entrée doit produire une empreinte de sortie radicalement différente de celle de l'entrée initiale, comme illustrée ci-dessous avec SHA-1. En effet, une autre application des fonctions de hachage est d'assurer l'intégrité d'un fichier. C'est pourquoi il n'est pas rare, lorsque vous téléchargez un logiciel, de trouver à côté du lien le *checksum*, ou somme de contrôle, qui est le *hash* du fichier d'origine. Si le fichier téléchargé produit une somme différente, c'est qu'il a été altéré et remplacé par une version pouvant contenir du code malveillant. **Figure 1**

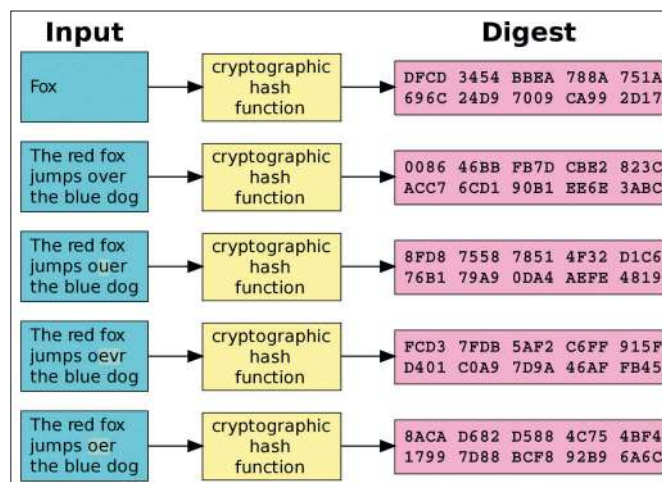


Figure 1



Sonia Seddiki

Développeuse depuis 5 ans, possédant un fort penchant pour le backend et l'infra. Globe-trotteuse compulsive avec une passion secrète pour la cybersécurité et une (encore plus secrète) pour les dépôts Git bien rangés.

Pour compléter le hachage, il est devenu commun de rajouter un salage, ou *salt* en anglais. Le sel est une chaîne de caractères apposée à la fin du mot de passe avant son hachage. Stocké en clair dans la base de données, son but n'est pas d'être secrète, mais de donner davantage de travail aux pirates¹. Une bonne pratique est de le générer aléatoirement pour chaque utilisateur. Ainsi, une fois le sel ajouté, deux mots de passe identiques produisent un *hash* différent.

Une troisième précaution peut venir compléter le hachage et le salage : l'ajout du *pepper*, le poivre. Contrairement au sel, le poivre est unique, secret et stocké hors de la base de données, le plus souvent dans le code source. Cela permet - en théorie - de fortement ralentir le *cracking* si l'attaquant n'a récupéré que la base de données et non le code source. Il existe plusieurs façons d'implémenter le *pepper*, et toutes ne se valent pas ! Mais nous en reparlerons plus tard.

Toutes ces précautions sont prises avec le même objectif en tête : ralentir les pirates !

En effet, nous avons établi précédemment que le hachage est irréversible. Une seule solution pour retrouver un mot de passe à partir de son *hash* : tester toutes les combinaisons de caractères possibles, jusqu'à trouver celle qui produira le même *hash*. C'est ce qu'on appelle une attaque par force brute, ou *bruteforce*.

La force brute, mais en finesse

Le maître mot lors d'un *bruteforce* est le nombre de combinaisons à tester. Pourquoi la plupart des sites demandent-ils un mot de passe de 8 caractères minimum, comprenant au moins une majuscule et un chiffre/un caractère spécial ? Pour augmenter le nombre de combinaisons possibles. Petit intermède mathématique :

Le nombre de combinaisons possibles de caractères pour une longueur de mot de passe donnée est (**longueur de l'alphabet**)^(longueur du mot de passe).

Pour vous en convaincre : considérez un mot de passe sous forme de code à 4 chiffres, chaque chiffre étant compris entre 0 et 9. Vous avez donc 10 chiffres possibles et 4 chiffres par code, soit $10^4 = 10\,000$ combinaisons possibles en tout. Or les codes possibles vont de 0000 à 9999, ce qui correspond bien aux 10 000 combinaisons annoncées.

Sur ce modèle, calculons le nombre de combinaisons possibles pour un mot de passe de 8 caractères, avec l'alphabet latin de 26 lettres, des chiffres de 0 à 9 et des caractères spéciaux parmi ()[]{}!;,:@_- (13 en tout) :

	Longueur de l'alphabet	Nombre de combinaisons
Minuscules	26	208 827 064 576
Minuscules, majuscules	$26 + 26 = 52$	53 459 728 531 456
Minuscules, majuscules, chiffres	$26 + 26 + 10 = 62$	218 340 105 584 896
Minuscules, majuscules, chiffres, caractères spéciaux	$26 + 26 + 10 + 13 = 75$	$\sim 1\,001\,129\,150\,000\,000$

Des chiffres qui donnent le vertige, et pourtant pas si insurmontable. Les logiciels de *cracking* peuvent exploiter la puissance des cartes graphiques afin de drastiquement

réduire les temps de calcul. Selon une analyse de 2017, si un *bruteforce* avec un processeur seul met 92 ans à parcourir les quelques 200 000 milliards de possibilités d'un mot de passe à 8 caractères avec majuscules, minuscules et chiffres, l'ajout d'une carte graphique réduit ce temps à 7 minutes².

Il est d'ailleurs possible de réduire davantage le nombre de combinaisons sur lesquelles itérer. Dans le dernier cas, les calculs précédents supposent que chaque caractère peut être une minuscule, une majuscule, un caractère spécial ou un chiffre, ce qui est rarement le cas. Une logique courante est de commencer par une majuscule et finir par un chiffre et un caractère spécial. Nous pouvons transcrire cette logique dans un *masque*.

Une attaque par masque consiste à affiner le *bruteforce* en variant l'alphabet possible suivant la place du caractère dans le mot de passe. Démonstration avec notre logique précédente :

Légende - X: majuscule, x: minuscule, 0: chiffre, %: spécial

	X	x	x	x	x	x	0	%
Nombre de caractères possibles	26	26	26	26	26	26	10	13
Nombre total de combinaisons	$26 \times 26 \times 26 \times 26 \times 26 \times 26 \times 10 \times 13 = 40\,159\,050\,880$							

40 milliards ! C'est environ 5000 fois moins que nos 200 000 milliards précédents, lesquels pouvaient déjà être listés en 7 minutes ! Nous pourrions pousser le vice plus loin : supposer que la plupart du temps, le chiffre et le caractère spécial utilisés à la fin sont 1 et !, et ne faire varier que les lettres, portant le nombre de combinaisons à tester à , soit environ 300 millions.

Continuons dans la réduction du champ des possibles. Il est rare que nos mots de passe soient une succession aléatoire de caractères, car ils seraient trop difficiles à retenir. Un mot de passe est plus souvent constitué de noms et de dates. Selon une étude menée par Google, 59 % des Américains incorporent le nom de leur animal de compagnie, de leurs enfants, voire le leur, lors du choix de leur mot de passe. Ce constat nous amène à notre deuxième arme de *cracking* : le dictionnaire.

Le dictionnaire, comme son nom le suggère, est une liste de mots. L'un des plus célèbres est *RockYou*, (qui a donné son nom à la récente liste *RockYou2021*), et qui était lui-même issu d'une liste de mots de passe. Plutôt que de s'en remettre à un *bruteforce* pur, on peut privilégier l'utilisation d'une telle liste couplée à un masque. Un exemple avec la liste d'origine, qui comptait environ 14 500 000 entrées :

Structure du mot de passe	Total de combinaisons
1 mot du dictionnaire	$14\,500\,000 \times (2021 - 1899)$
+ 1 année de naissance entre 1900 et 2021	$= 1\,769\,000\,000$

(1) Pirate, ou cracker, et non hacker. Bien que la culture populaire et les médias fassent souvent l'amalgame, ce sont deux catégories d'individus que tout oppose. "The basic difference is this: hackers build things, crackers break them."

(2) <https://tcblog.protiviti.com/2017/09/26/the-8-character-password-is-dead/>

L'attaque par dictionnaire est bien plus efficace, car plus ciblée, et avec des mots parfois plus longs que 6 ou 8 caractères. Elle permet donc de cracker des mots de passe de plus de 10 caractères, là où un *bruteforce* pur serait bien trop long : en supposant que l'on peut calculer 100 milliards de combinaisons par seconde³, il nous faudrait plus de 19 ans ! Passez à 11 caractères, et il vous faudra presque 19 siècles⁴. Pour parachever le tout, les logiciels de *cracking* permettent également de fournir un fichier de "règles" afin de préciser certaines variations spécifiques de caractères. Avec un dictionnaire qui ne connaît que `password`, le masque précédent ne testerait que de `password0000` jusqu'à `password9999`, et passerait à côté d'un mot de passe tel que `Pa$$w0rd2021`. Il est désormais possible de spécifier des règles de substitution habituelles, telles que remplacer `o` par `0` ou `s` par `$`, permettant d'optimiser au mieux le nombre de tests à faire.

On prend les mêmes, et on recommence - Les *rainbow tables*

Les techniques ci-dessus mettaient l'accent sur le *cracking* d'un mot de passe en particulier. Comment l'optimiser à l'échelle de toute une base de données ? Une pratique courante est de créer des *rainbow tables*, des listes de *hashes* précalculées. Par exemple, vous pourriez stocker toutes les empreintes calculées pour le premier mot de passe dans une map `<password, hash>`, et regarder si les *hashes* des autres mots de passe de la base se trouvent dans cette map. Il s'agit ici d'échanger de la complexité temporelle (le temps de calcul du *hash*), contre de la complexité spatiale (l'espace disque requis pour stocker la *rainbow table*) : interroger une map est quasi instantané et représente un gain de temps considérable.

Quelles solutions s'offrent à nous ?

Rassurez-vous, tout n'est pas perdu. Côté fournisseur de service, l'ajout d'un *salt* individuel est efficace contre les *rainbow tables*. Si chaque mot de passe est haché avec un sel différent, l'attaquant ne pourra pas réutiliser les tables des mots de passe précédents et sera obligé de recalculer toutes les empreintes à chaque nouveau mot de passe.

De même, l'ajout de *pepper* compliquera davantage les choses. Mais attention, une mauvaise implémentation et nous pourrions vite nous tirer une balle dans le pied !

Une première approche triviale serait de rajouter le *pepper* au *salt* et au mot de passe, afin de rallonger la chaîne de caractères à hacher. Étant donné qu'il est inconnu de l'attaquant, il ajoute une quantité massive de possibilités. Avant de pouvoir retrouver ne serait-ce qu'un mot de passe, il faudrait tester toutes les versions possibles de *pepper*- et il sera difficile de faire une supposition sur sa longueur et sa nature, il n'y en a qu'un et il est stocké, donc pas de compromis à faire sur sa complexité.

Premier problème : certaines fonctions de hachage acceptent un nombre très limité de caractères, et une combinaison (`pepper.salt.password`) pourrait vite dépasser cette limite. Ce n'est

pas une limitation technique, mais un véritable choix de sécurité. En effet, le temps de calcul d'une fonction de hachage dépend aussi de la chaîne qui lui est fournie en entrée, et passer de très longues chaînes de manière répétée à une application pourrait causer un déni de service, car son backend serait monopolisé par leur hachage⁵.

Deuxième problème : il est impératif de pouvoir facilement opérer une rotation des secrets en cas de fuite, autrement dit ici pouvoir changer le *pepper* s'il était découvert. Or, si l'on stocke `hash(pepper.salt.password)` dans la base de données, cela devient impossible à moins de régénérer tous les mots de passe !

Une approche plus recommandée aujourd'hui est de se servir du *pepper* comme clé de chiffrement afin de générer un HMAC (*keyed-Hash Message Authentication Code*) ou de chiffrer le *hash* (le chiffrement étant réversible, il est possible de le déchiffrer et le chiffrer avec la nouvelle clé en cas de fuite)⁶. Dans tous les cas, il n'y a pas de solution simple et l'ajout d'un *pepper* implique des coûts de maintenance plus élevés (un secret supplémentaire à protéger et plus de travail en cas de fuite).

Toutefois, le plus important reste de se reposer sur des fonctions de hachage déjà implémentées ! La science du hachage est complexe, et vouloir implémenter sa propre fonction (ou combiner plusieurs fonctions de hachage connues) peut s'avérer désastreux pour la sécurité de vos mots de passe. De nos jours le standard est la fonction `bcrypt`, disponible en bibliothèque dans la plupart des langages.

Côté utilisateur, la règle d'or à respecter lors du choix de votre mot de passe : considérer que son *hash* sera inévitablement récupéré. Ne comptez pas sur une sécurité infaillible des services auxquels vous vous inscrivez. Votre objectif est d'être dans la partie "non-rentable" de la liste, autrement dit les *hashes* trop complexes à pirater au regard de leur "valeur". Voici donc quelques astuces afin de peaufiner votre stratégie :

- Entrez votre mot de passe sur Pwned Passwords pour vérifier s'il a déjà été compromis, et changez-le le cas échéant
- Ajoutez de l'authentification multifactorielle quand elle est disponible
- Choisissez un mot de passe plus robuste : privilégiez des passphrases plutôt que des passwords, sortez des logiques habituelles (commencer par une majuscule et finir par un chiffre), et n'hésitez pas à créer des mots de passe à usage unique pour des sites que vous ne fréquentez que ponctuellement
- Utilisez un gestionnaire de mots de passe

(5) Le framework Python Django en a d'ailleurs fait les frais il y a quelques années : <https://www.djangoproject.com/weblog/2013/sep/15/security/>

(6) https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#peppering

(3) Réalisable avec une NVidia 2080Ti (non overclockée), carte graphique assez récente : <https://twitter.com/hashcat/status/1095807014079512579?s=20>

(4) Source : <https://www.grc.com/haystack.htm>

Sources

https://fr.wikipedia.org/wiki/Fonction_de_hachage_cryptographique#/media/Fichier:Cryptographic_Hash_Function.svg
<https://storage.googleapis.com/gweb-uniblog-publish-prod/documents/PasswordCheckup-HarrisPoll-InfographicFINAL.pdf>



**Cristina
Hernando Pajuelo**

Ingénieure Télécoms avec un Master en Marketing, j'accompagne mes clients sur leurs problématiques Google Ads, Tracking Google Tag Manager et Landing Page.

Je les aide à augmenter la qualité et le nombre de prospects qui visitent leur site Internet avec un objectif simple et direct : faire évoluer le CA de leur activité par le biais de Google Ads.

Touche pas à mon code !

Avec cet article vous saurez comment votre équipe de développement peut être déchargée de quelques codages, en toute sécurité grâce à Google Tag Manager.

La pandémie a mis en évidence qu'avoir un site en ligne est une première étape de survie pour un business, mais loin d'être suffisante. Vous devez, en plus, amener du trafic qualifié vers votre site et savoir l'exploiter pour convertir ces visiteurs en prospects, et ce qui est mieux encore, en clients.

Votre site doit être assez performant comme pour retenir l'attention de vos visites, leur faire remplir un formulaire de contact, passer commande, etc. Alors, comment savoir si votre site tire le meilleur parti de ces visites ? Pour analyser leur comportement, il faut avoir installé au préalable un niveau de suivi 'tracking' suffisant.

Quel suivi ou tracking est implémenté dans 80% des entreprises ?

Votre équipe de développement de votre site vous a assuré qu'Analytics a été installé. Vous pouvez étudier jusqu'à présent le taux de rebond ou la durée de session, ces métriques qu'Analytics fournit par défaut. C'est un début, mais vous passez à côté des données qui vont être déterminantes pour la survie de votre business. Ne blâmez pas vos développeurs s'ils s'en sont contentés jusqu'à présent. Ils ont déjà beaucoup à gérer entre les mises à jour, les corrections de bugs ou la sortie de la version suivante.

Les seules personnes qui connaissent les données dont ils ont besoin pour améliorer la performance de votre site est l'équipe marketing. L'équipe marketing n'a pas, normalement, des compétences de codage ni l'accès pour le faire : ils ne devraient pas toucher au code ! Et, pourtant...

Des startups françaises qui s'en sortent très bien telles que ManoMano, Opisto ou Nannybag utilisent à ce jour un outil gratuit et incontournable : **Google Tag Manager**.

Toutes ces entreprises ont un point commun : elles ont compris l'importance d'un site web qui convertit les visites en prospects. Pour ce faire, elles suivent efficacement leur activité de marketing sans déranger l'équipe de

développement tous les quatre matins, déjà occupés avec des hot fix, des évolutions de la version actuelle, etc.

Du tracking marketing n'est pas un incident en production. Cependant, le manque de réactivité à implémenter des codes de tracking peut faire perdre de l'argent, beaucoup même, suite à des conclusions et décisions erronées sur les campagnes de publicité en cours.

Quelles raisons devraient vous pousser à installer Google Tag Manager dès maintenant ?

Pour commencer à utiliser Google Tag Manager, rien de plus simple que d'ouvrir un compte gratuit et faire installer le code (conteneur) par un.e développeur.se dans votre site, une seule fois.

Ensuite, tout est paramétré depuis l'interface Google Tag Manager par l'administrateur (nous y reviendrons) : le code, le déclenchement, ou encore le niveau de données à retourner à Google Analytics.

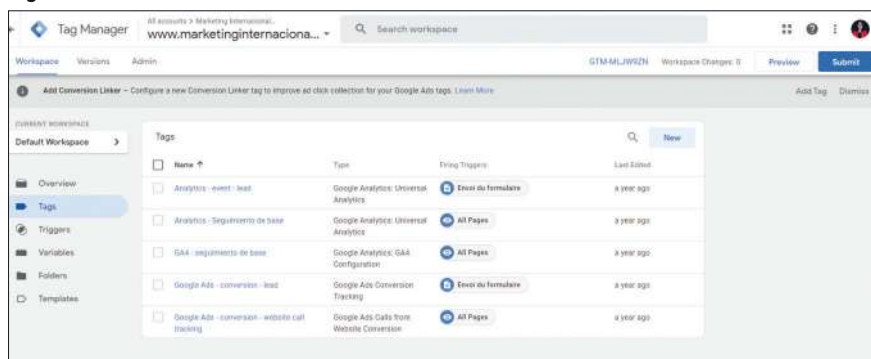
Quelques avantages de l'outil :

- 1 Installer facilement le suivi des plateformes tiers, telles que Google Ads, Facebook Ads, LinkedIn Ads, et même Google Analytics. L'information arrive en temps réel sur les plateformes publicitaires, qui utilisent l'information pour mieux cibler l'audience.
- 2 Suivi avancé de tous les clics, ou des clics sur un CTA précis, envoi de formulaire, montant de la commande, etc. Vous imaginez la différence entre mesurer les résultats d'une campagne par le nombre de visiteurs uniques que par le montant dépensé par ces visiteurs.
- 3 Tous les codes se retrouvent dans une même interface, et non, éparpillés par votre site, difficiles de 'troubleshooter' en cas de problème. **Figure 1**
- 4 Il s'intègre aux autres outils de Google.
- 5 Gain de temps et de réactivité pour toutes les équipes.

À titre d'exemple, un de mes clients m'a délégué l'implémentation Google Tag Manager pour réussir à faire remonter les chiffres de CA dans Analytics. Grâce à ça, nous avons engrangé +300% de conversions dès le premier mois au même moment que nous avons économisé environ 70% de charge de son équipe IT sur les demandes de marketing.

Un autre client ne voulait surtout pas nous laisser travailler sur Google Tag Manager. Il voulait garder la main sur l'implémentation du code. Malheureusement, il n'a pas réussi à faire implémenter les codes de suivi avancés, nécessaires

Figure 1



pour envoyer des données pertinentes pour améliorer l'IA de Google Ads. En conclusion : cela lui a fait perdre du temps, de l'argent et l'avantage sur ses concurrents.

Vous êtes convaincus ? Installez Google Tag Manager en un clin d'œil

Pour pouvoir installer Google Tag Manager, vous devrez toucher au code. On vous aura menti ? Non, c'est le seul moment où vous aurez à le faire. Ensuite, l'administrateur Google Tag Manager fera tout depuis l'interface GTM, sans toucher à votre code directement.

Si bien qu'il faut avoir un niveau avancé en tracking pour pouvoir bien travailler sur Google Tag Manager, la création de votre compte ainsi que l'installation sur votre site ne sont pas sorciers. **Figures 2 et 3**

Ensuite, vous n'avez qu'à installer le code fourni dans votre code. **Figure 4**

Quel niveau de suivi je peux implémenter ? Exemples concrets

- 1 Installer le suivi Analytics de base sur tout le site
- 2 Tracker les clics sur des liens
- 3 Suivre les clics sur un bouton spécifique (même lorsqu'il n'y a pas de lien associé)
- 4 Tracker le nombre d'appels en cliquant sur le téléphone de votre site
- 5 Le nombre de formulaires envoyés
- 6 Téléchargement de documents : brochures, e-books, etc.
- 7 Faire remonter dans Analytics le montant d'une transaction
- 8 Visites à des pages spécifiques, tel que la page de remerciement
- 9 Installer des Pixels de plateformes tiers tel que Facebook Ads, Google Ads, LinkedIn Ads
- 10 Mettre à jour des tags existants pour qu'ils soient en norme par rapport au RGPD
- 11 Et bien des autres

Cas pratique. Installer le suivi de tous les clics sur votre site.

Voici comment implémenter un suivi assez élémentaire, mais nécessaire, pour tracer tous les clics qui ont lieu sur votre site.

- 1 Depuis votre conteneur GTM, créez un nouveau déclencheur dit "Trigger". Nommez le "All clicks" et sélectionnez un type de déclencheur par défaut tel que : "Click > All elements".
 Spoiler : si vous choisissez "Just links" vous ne pourrez tracer que les clics sur des liens. **Figure 5**

- 2 Enregistrez votre nouveau déclencheur. **Figure 6**

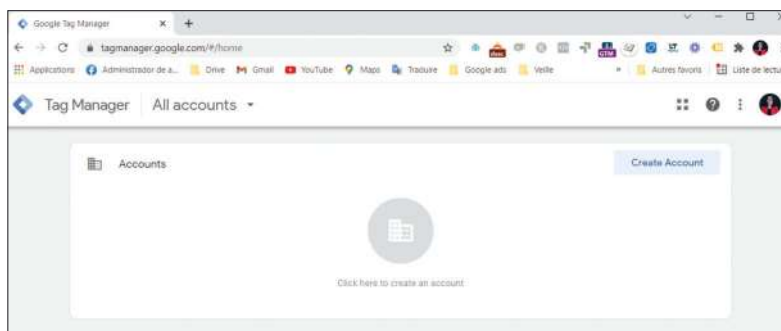


Figure 2

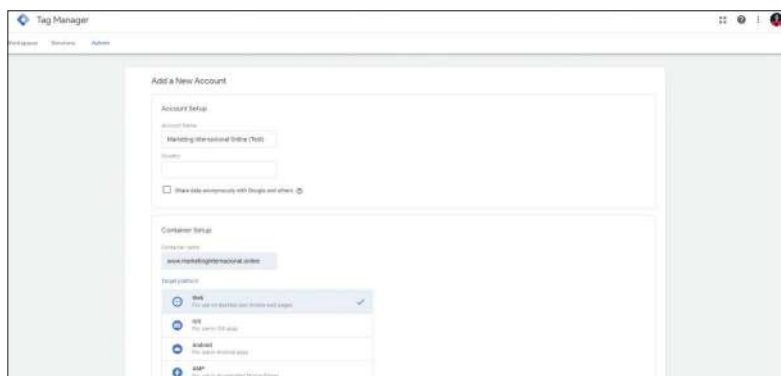


Figure 3

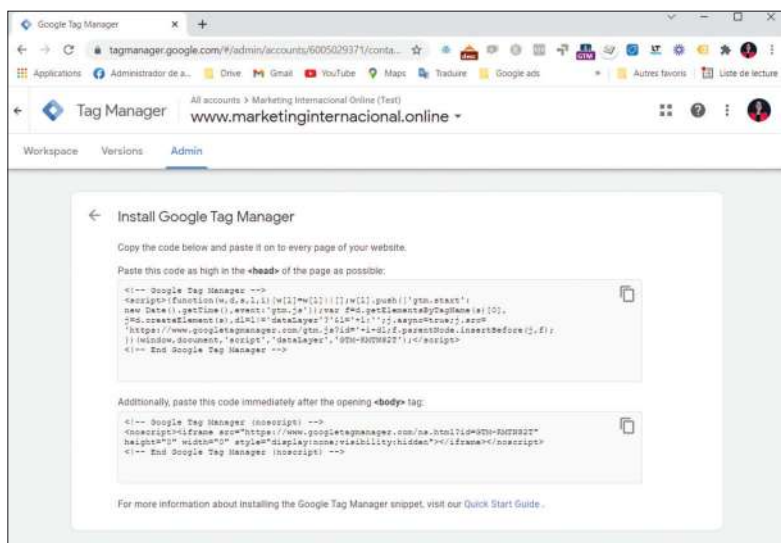


Figure 4

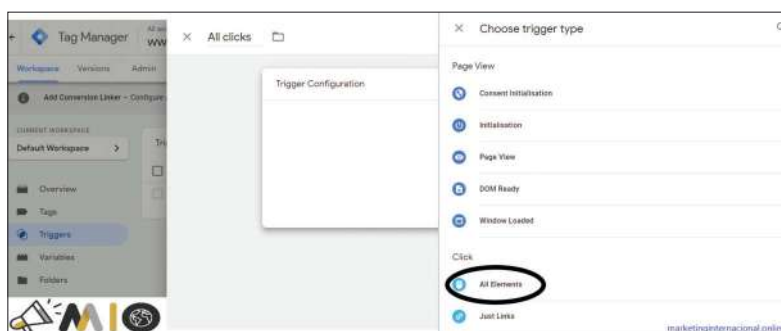


Figure 5

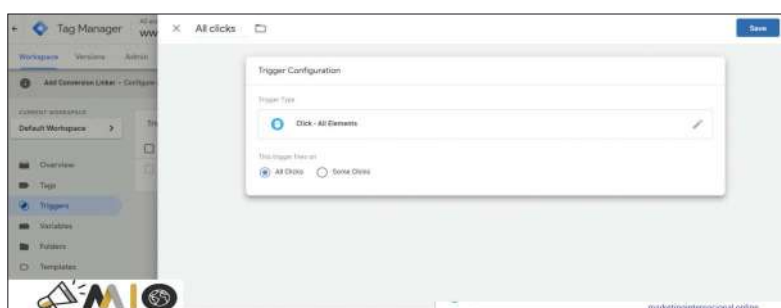


Figure 6

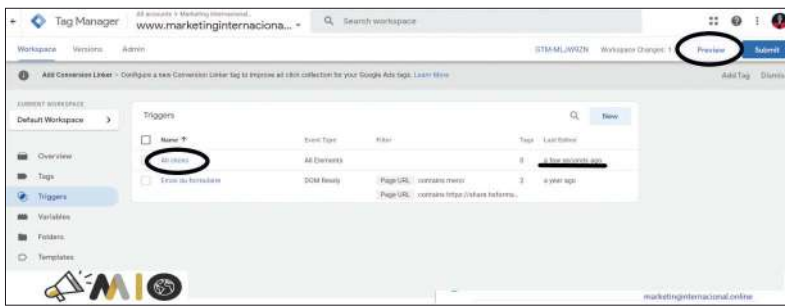


Figure 7

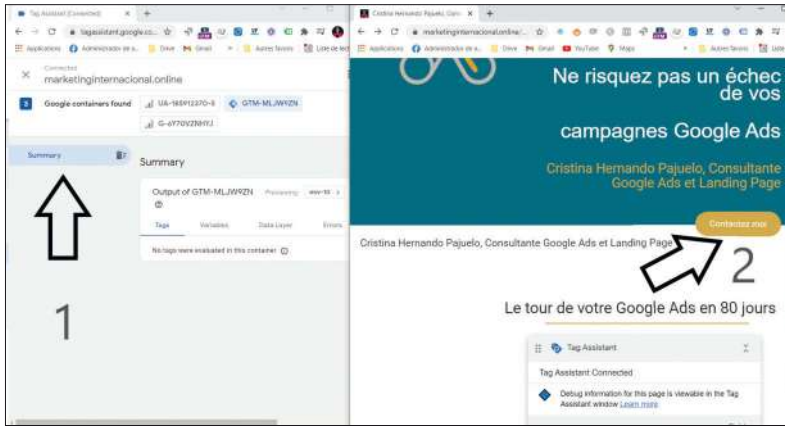


Figure 8

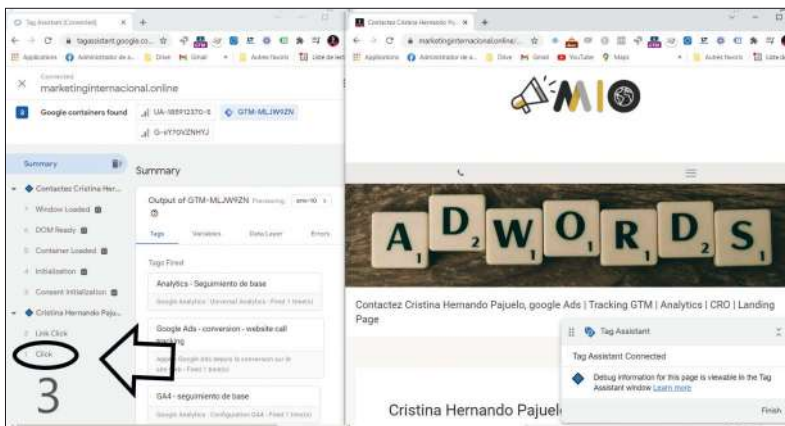


Figure 9

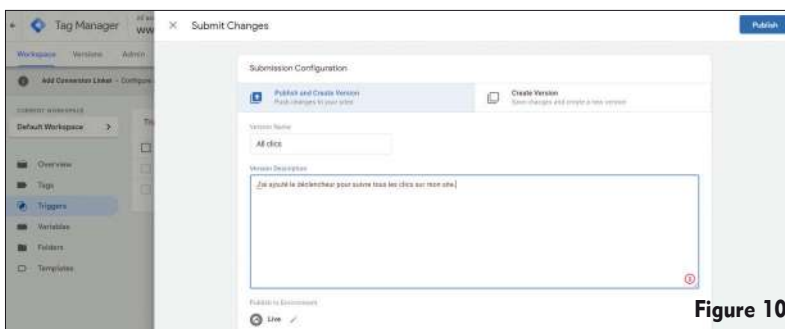


Figure 10

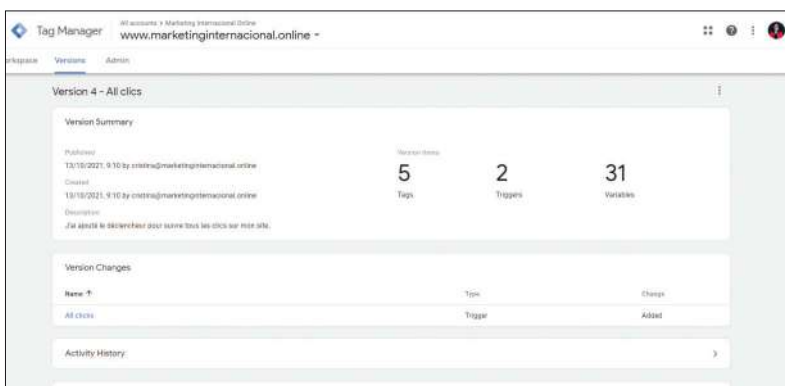


Figure 11

3 Faites un test grâce à l'outil de prévisualisation. (Preview Mode) **Figure 7**

4 Cliquez un bouton de votre site
Voici ce qui se passe dans votre interface GTM (preview) :

- 4.1 Aucun événement
- 4.2. Clic sur le bouton 'Contactez-moi'
- 4.3. Le clic est tracé et j'arrive sur la page en question **Figures 8 et 9**

5 Une fois que vous êtes satisfait de la configuration, vous n'avez qu'à envoyer la modification en production. Notez ici que vous pouvez faire des modifications dans l'interface, les tester et les envoyer en production que lorsque vous êtes satisfaits du résultat. **Figures 10 et 11**

Ceci n'est qu'une configuration très élémentaire. Voici une liste plus exhaustive des tags à installer sur votre site via <https://www.marketinginternacional.online/blog/articles/google-tag-manager-comment-l-utiliser-pour-configurer-google-analytics-et-google-ads-tutoriel-complet>

Déléguer l'implémentation du suivi de tous les événements sans déranger un développeur... le rêve ou un risque ?

Introduire un bout de code sur votre site, même via Google Tag Manager n'est pas un geste anodin. Vous pouvez ralentir même faire tomber un site si l'administrateur GTM ne fait pas son travail correctement.

Si GTM s'est répandu comme une traînée de poudre, c'est parce que ce type d'incidents n'arrive presque jamais et voici pourquoi :

- Possibilité d'activer, modifier ou enlever les codes rapidement.
- Dans votre conteneur GTM vous trouverez des balises existantes prédéfinies. Elles ont déjà été testées, et cela se traduit en moins de risque.
- L'administrateur a la possibilité de prévisualiser le tag (balises) configuré avant de le mettre en production.
- Pour que les tags soient en production, il faut faire une action manuelle : les publier.
- Roll back facile : à chaque publication, une nouvelle version est enregistrée : avec la modification réalisée, l'utilisateur qui l'a faite, et la date. Il est extrêmement facile et rapide de revenir en arrière en cas d'impact sur le site.
- Si cela vous rassure, instaurer un système de gouvernance à l'intérieur de votre conteneur : une personne compétente peut superviser et approuver une modification.

Conclusion

Vous l'aurez compris, déléguer l'implémentation d'un bon nombre de scripts à un administrateur Google Tag Manager comporte zéro risque. Vous pouvez gagner du temps pour vos développeurs tout en améliorant de façon très rapide et simple l'information sur le comportement de vos visiteurs.

Pourquoi j'ai voulu faire de l'accessibilité

L'accessibilité numérique est une obligation prévue par un texte de loi de 2005, pour l'égalité des droits et des chances, édité par la direction interministérielle du numérique, le RGAA (Référentiel général d'amélioration de l'accessibilité), or aujourd'hui encore l'accessibilité demeure le parent pauvre des politiques publiques.

Personnes handicapées ou personnes en situation de handicap ?

J'ai souvent entendu : "l'accessibilité c'est faire des sites pour les handicapés", cette définition bien que très réductrice trouve tout son sens quand dans certains documents officiels on parle de "personnes handicapées" comme la loi assurant l'exercice des droits des personnes handicapées en vue de leur intégration scolaire, professionnelle et sociale par exemple.

Ce qui est vraiment dérangeant je trouve, c'est que l'on définit quelqu'un par rapport à son handicap, c'est-à-dire que c'est inhérent à sa personne, au même titre qu'un trait de caractère ou une personnalité. Le handicap c'est avant tout une **situation**, c'est pourquoi je parle toujours de **situation de handicap**. Je considère que tout le monde prend ses responsabilités, c'est le manque d'adaptation de l'environnement et l'incapacité de réaliser les activités de la vie quotidienne qui crée une situation de handicap.

L'accessibilité une pratique à part entière

Mais en fait, l'accessibilité numérique c'est quoi ? On pourrait le définir comme le fait de "Permettre à des personnes en situation de handicap d'accéder aux contenus et services numérique.", cette définition est simple et claire, cependant j'aimerais la compléter en y ajoutant une dimension liée spécifiquement au métier de développeur. Pour ma part j'aime voir l'accessibilité comme une facette de l'artisanat du logiciel ou software craftsmanship, c'est-à-dire un savoir-faire, un ensemble de bonnes pratiques, qui mettraient l'accent sur la qualité du code avant tout.

On ne fait pas du code accessible parce que des lois nous y obligent, mais parce qu'en ne le faisant pas on va tout simplement à l'encontre des fondements du World Wide Web et des bonnes pratiques de développement Web en général.

Comment expliquer qu'avec le nombre de nouveaux développeurs qui intègrent le marché du travail chaque année, l'inaccessibilité ne soit pas un sujet en voie d'extinction ? Après tout, l'objectif principal du World Wide Web Consortium c'était de "développer un monde où tous les gens peuvent utiliser le Web pour communiquer, collaborer et innover librement", cela paraît bien utopique quand on sait que moins de 20% des sites en français sont accessibles...

Un manifeste accessible ?

C'est en relisant le Manifeste Agile que je me suis posé cette question : c'est un document rédigé en 2001, qui a été le fruit de la réunion de 17 experts du développement logiciel,

l'objectif était de définir une méthode unificatrice afin de tenir les délais tout en tenant également le budget lors de la réalisation d'un projet.

On lie le développement aux besoins de l'entreprise, le changement est le bienvenu et la satisfaction du client au centre des préoccupations.

Pourquoi ne pas créer un document qui mettrait en avant des valeurs propres à l'accessibilité et qui donnerait aux développeurs une direction commune, une éthique de travail ?

Un des grands problèmes qui explique le manque d'accessibilité, aujourd'hui selon moi c'est l'effort fourni dans le choix des outils plutôt que dans la réalisation du projet en lui-même. L'utilisateur n'est plus au centre des préoccupations ce qui est inconcevable, encore plus quand notre métier c'est de créer des interfaces pour de vraies personnes qui n'ont pas toutes accès de la même façon à l'information.

Pour ces personnes, le contenu de votre site compte beaucoup plus que les outils que vous utilisez pour le développer.

La vraie performance, ce n'est pas d'avoir un super projet réalisé avec les derniers frameworks ou bibliothèques du moment, mais c'est de créer un projet utilisable par n'importe qui. L'accessibilité ne devrait pas être un choix, à partir du moment où vous décidez de créer des interfaces, vous signez implicitement un contrat qui vous oblige à faire tout votre possible pour permettre au plus grand nombre d'y avoir accès.

Le problème de la désinformation

Comme de nombreux développeurs, j'ai appris mon métier via une formation rapide. En 9 mois j'ai découvert les bases de la programmation front-end, HTML, CSS et JavaScript. Puis, rapidement j'ai commencé à m'intéresser aux frameworks, à l'époque React était encore en version bêta et AngularJS était le framework à utiliser pour son projet. J'avais vaguement entendu parler d'accessibilité, mais ça ne semblait pas indispensable dans ce que je voulais faire. Pour moi l'accessibilité était vraiment une discipline à part, qui n'avait rien à voir avec ce que je voulais faire en entreprise. Moi je voulais simplement faire des interfaces, et pas me spécialiser dans un domaine qui m'était complètement inconnu.

Mais à cette époque je ne comprenais pas que les deux étaient très liés. L'accessibilité ne devrait pas être perçue comme quelque chose d'ennuyeux à mettre en place, malheureusement dans beaucoup de projets, très peu de personnes connaissent les normes d'accessibilité ou bien ont bénéficié d'une quelconque formation à ce sujet.

Dans bien des projets, il n'existe qu'un référent par défaut : c'est une personne qui a montré son intérêt pour le sujet et



Prescillia Seck Ndedi

Consultante fullstack chez Zenika.

Je donne régulièrement des talks sur l'accessibilité, le management, la tech



qui se retrouve à devoir toutes les revues de codes en suivant les normes accessibles, car personne d'autre ne souhaite le faire, ce qui est très dommage.

Produire du code qui a du sens

Pendant très longtemps j'ai beaucoup négligé les bases, je m'en rends compte maintenant, le HTML par exemple, est un langage de balisage qui sert à structurer du texte. À part créer la structure d'une page, c'est-à-dire mettre des titres, des images... je ne voyais vraiment pas ce qu'il y avait de plus à apprendre.

Comme beaucoup, j'ai surexploité l'utilisation des balises `div` et `span`, car c'était pratique, sans me poser la question de savoir si c'était bien ou pas, ça fonctionnait parfaitement comme ça et je l'avais souvent vu dans des projets en entreprise. Je n'avais donc aucune raison de m'en soucier.

Le HTML sert à structurer du texte oui, mais il sert surtout à lui donner du sens et ce n'est que très tardivement que j'ai découvert et compris l'utilité de la **sémantique**. Faire de la sémantique c'est se concentrer sur le fond, la signification des informations contenues dans les pages Web plutôt que l'apparence. L'objectif étant de permettre au navigateur de restituer correctement les écrans afin que des personnes qui ne peuvent pas les voir par exemple puissent quand même avoir accès au contenu de vos pages grâce aux lecteurs d'écran.

Ça peut paraître futile et sans intérêt, pour ma part cela m'a beaucoup dérangé, car je me suis rendu compte que depuis tout ce temps j'étais dans le faux, que mon code n'avait strictement aucun sens.

Je me suis rendu compte qu'en négligeant la sémantique et les bonnes pratiques d'accessibilité, je ne connaissais pas les bases de mon propre métier. Faire de l'accessibilité ce n'est pas juste suivre des pratiques parce que l'on est obligé, c'est aussi savoir utiliser les bonnes choses au bon endroit. Un manque d'accessibilité démontre surtout un manque de maîtrise des bases du développement Web.

Un petit aide mémoire pour faire de l'accessibilité

Si vous trouvez que faire de l'accessibilité c'est trop compliqué, ou que vous ne savez pas par où commencer. Je vous ai fait une petite liste qui aborde les points importants de l'accessibilité Web.

On ne vous demandera jamais de faire 100% d'accessibilité, car c'est tout simplement impossible, mais une accumulation de petites choses peut avoir de grands effets. Faites au plus simple, commencez par des petites choses, essayez de comprendre en quoi c'est une bonne pratique, cela vous aidera à développer un "réflexe accessible".

Utiliser les bonnes balises aux bons endroits

J'en ai parlé plus haut, la sémantique est indispensable quand vous faites de l'accessibilité, faites au plus simple quand vous en avez la possibilité. Il vaut mieux une structure qui respecte la sémantique plutôt qu'une structure complexe que les lecteurs d'écran ne pourront pas restituer.

Évitez l'utilisation des balises `div` et `span` le plus possible, il existe déjà de nombreuses balises structurales :

- `<header>` pour vos contenus introductifs

- `<nav>` pour la navigation dans un document
- `<main>` pour le contenu majoritaire dans la page
- `<article>` pour les blocs autonomes
- `<section>` pour une section générale d'un document
- `<aside>` pour une partie d'un document dont le contenu n'a qu'un rapport indirect avec le contenu principal du document
- `<footer>` pour les pieds de page

Figure 1

L'idée c'est de hiérarchiser la structure de vos pages afin de permettre aux lecteurs d'écran de les restituer correctement. La sémantique n'est pas une science exacte, je prends l'exemple des balises `section` et `article` qui sont souvent échangés, soyez simplement cohérent dans la structure de vos pages.

Les titres de pages

Ils se situent dans la partie *head* et sont définis via la balise `title`. Ils s'affichent à différents endroits : navigateurs, favoris... l'objectif est de permettre un meilleur référencement et une meilleure accessibilité à vos contenus par les utilisateurs qui pourront plus facilement identifier votre site.

Les titres de section

Utilisez les titres et sous-titres à l'aide des éléments HTML de `h1` à `h6`, ne dupliquez pas les titres.

Les niveaux doivent être logiques aussi veillez à ce que la structure ne comporte pas de "trous", un titre `h2` ne peut pas être suivi d'un titre `h5`.

Dans la mesure du possible, faites en sorte que chaque page comporte un titre de niveau 1. C'est très utile pour les navigateurs, moteurs de recherche et outils de vocalisation qui vont restituer la structure d'une page, les utilisateurs peuvent plus facilement accéder au contenu qui les intéresse.

Évitez les problèmes d'affichage

L'UTF-8 est le codage de caractères universels le plus utilisé, il permet d'afficher de nombreux caractères. Cela vous permettra de prévenir de potentielles erreurs d'affichages.

L'information n'est pas véhiculée uniquement par la couleur

Comment une personne limitée dans la perception des couleurs peut accéder à une information si le seul indicateur est la couleur ? Il est donc important de fournir un complément à la couleur par exemple un balisage sémantique (`strong`, `em` ...) ou bien une mention textuelle comme un astérisque.

Contraster sa page

Ce point n'est pas réservé qu'aux utilisateurs en situation de handicap, une police couleur jaune sur un fond blanc est très difficilement lisible pour n'importe quel utilisateur. Il faut donc veiller à respecter un ratio de contraste minimal de 3:1.

Figure 2

La navigation au clavier

Il existe de nombreuses façons d'utiliser internet : les commandes vocales, claviers, *eye-tracking*, etc. Il est donc indispensable de fournir une alternative à l'utilisation classique de la souris. Le focus clavier va signaler ou se trouve le curseur

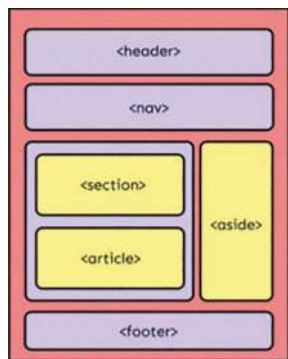


Figure 1
Une page qui utilise le HTML sémantique

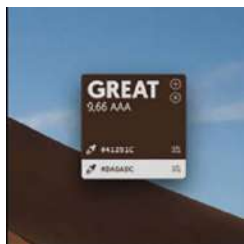


Figure 2
Au dessus une image avec un ratio de 9:66 et dessous une image avec un ratio de 1:98

dans une page, il est utilisable via la touche Tabulation du clavier. Il est donc important de ne pas masquer ou supprimer l'effet visuel de prise de focus. L'utilisateur doit pouvoir se repérer dans votre page et savoir où il se trouve.

Les alternatives textuelles

L'idée c'est de restituer à l'utilisateur des éléments qu'on ne peut que voir, assurez-vous que vos images possèdent une alternative textuelle, utilisez l'attribut "alt", c'est un texte alternatif qui permet d'ajouter une description visuelle, à vos images, pictogrammes... Soyez le plus précis possible, mais ne détaillez que si cela est nécessaire. Sans attribut, un outil d'assistance restituera le contenu de l'attribut "src" et une image "image1.jpeg" n'apporte aucune information pertinente à l'utilisateur.

Se documenter

Ce n'est qu'un petit échantillon de bonnes pratiques qui vous aideront à faire un pas vers l'accessibilité, il existe beaucoup d'autres ressources, articles que vous trouverez facilement sur la toile et qui vous permettront de faire de l'accessibilité.

Je vous partage les ressources que j'utilise le plus au quotidien :

- Documentation de WCAG
- Documentation du RGAA
- La checklist Qualité Web d'Opquast

WCAG

Web Content Accessibility Guidelines : il s'agit de standards publiés par le W3C (World Wide Web Consortium) qui vous aideront à rendre vos contenus Web plus accessibles aux personnes en situation de handicap, mais pas seulement. Elles vous permettront de faciliter de façon générale leur utilisation. Il y a une douzaine de règles organisées autour de 4 grands principes : perceptible, utilisable, compréhensible et robuste. Il existe trois niveaux de conformité :

- A : niveau de conformité minimal
- AA : niveau d'accessibilité amélioré
- AAA : niveau d'accessibilité adapté à certains contextes

Le niveau AA est le niveau recommandé par le W3C et les WCAG comme objectif de conformité, d'ailleurs WCAG rappelle que **"il n'est pas recommandé de se fixer le niveau AAA comme objectif à l'échelle de sites entiers, car il n'est pas possible de satisfaire à tous les critères de succès du niveau AAA pour certains contenus"**. Vous trouverez différents niveaux de lecture : les grands principes, les règles générales, les critères

de succès testables, des techniques de réussite et d'échec ainsi que des liens vers des ressources complémentaires. **Figure 3**

RGAA

Pour Référentiel Général d'Amélioration de l'Accessibilité, c'est propre à la France c'est une méthode d'application de la norme WCAG. Elle se présente en 2 parties, la première est réservée aux obligations à respecter et la deuxième est plus orientée professionnels du Web. **Figure 4**

Checklist Opquast

Opquast est une entreprise dont l'objectif est de créer des référentiels de bonnes pratiques, mais aussi de former des professionnels à l'accessibilité. Ils ont créé une checklist qui regroupe 240 bonnes pratiques pour améliorer l'accessibilité des sites Web. Leur interface est très simple et facile à lire c'est un bon point de départ quand on souhaite se documenter. **Figure 5**

Conclusion

L'accessibilité c'est permettre aux personnes en situation de handicap d'accéder aux contenus et services digitaux, mais c'est avant tout réduire les inégalités entre les personnes. C'est s'assurer d'être à la fois dans la **légalité** et dans l'**égalité**. En qualité de développeur nous sommes les acteurs majeurs de cette transformation et c'est à nous de prendre nos responsabilités et de tendre vers un Web plus juste pour tous.

Alors, on s'y met ?

Figure 3

La règle 1.1 de WCAG 2.0

Principe 1 : perceptible - L'information et les composants de l'interface utilisateur doivent être présentés à l'utilisateur de façon à ce qu'il puisse les percevoir.

Règle 1.1 Les équivalents textuels : proposer des équivalents textuels à tout contenu non textuel qui pourra alors être présenté sous d'autres formes selon les besoins de l'utilisateur : grands caractères, braille, synthèse vocale, symboles ou langage simplifié. *Commentaire à la règle 1.1.1 et 1.1.2*

1.1.1 Contenu non textuel : tout contenu non textuel présenté à l'utilisateur a un équivalent textuel qui remplit une fonction équivalente sauf dans les situations énumérées ci-dessous. (Niveau A)

- **Composant d'interface ou de saisie :** si le contenu non textuel est un composant d'interface ou s'il permet la saisie d'informations par l'utilisateur, alors il a un nom qui décrit sa fonction. (Se référer à la [règle 4.1](#) pour des exigences supplémentaires à propos des composants d'interface utilisateur ou des contenus qui permettent la saisie d'informations par l'utilisateur.)
- **Média temporel :** si le contenu non textuel est un média temporel, alors l'équivalent textuel fournit au moins une identification descriptive du contenu non textuel. (Se référer à la [règle 3.2](#) pour des exigences supplémentaires concernant les médias temporels.)
- **Texte :** si le contenu non textuel est un texte ou un exercice qui serait évalué s'il était présenté en texte, alors l'équivalent textuel fournit au moins une identification descriptive du contenu non textuel.
- **Contenu sensoriel :** si le contenu non textuel vise d'abord à créer une expérience sensorielle spécifique, l'équivalent textuel fournit au moins une identification descriptive de ce contenu non textuel.
- **CAPTCHA :** si ce contenu non textuel vise à confirmer que le contenu est consulté par une personne plutôt que par un ordinateur, alors un équivalent textuel est fourni pour identifier et décrire la fonction du contenu non textuel, et des formes alternatives de CAPTCHA sont proposées pour différents types de perception sensorielle afin d'accommoder différents types de limitations fonctionnelles.
- **Décoration, formatage, invisible :** si le contenu non textuel est purement décoratif, s'il est utilisé seulement pour un formatage visuel ou s'il n'est pas présenté à l'utilisateur, alors il est implémenté de manière à être ignoré par les technologies d'assistance.

Comment satisfaire à 1.1.1 (en anglais) - Commentaire 1.1.1.1 (en anglais)

Commentaire à la règle 1.1.1 en français

Les critères de succès

Explications des règles d'accessibilité

Thématisées

1. Images 2. Cadres 3. Couleurs 4. Multimédia 5. Tableaux 6. Liens 7. Scripts 8. Éléments obligatoires 9. Structuration de l'information 10. Présentation de l'information 11. Formulaires 12. Navigation 13. Consultation

1. Images

Critère 1.1. Chaque image porteuse d'information a-t-elle une alternative textuelle ?

Test 1.1.1 :

- Chaque image (balise ou balise possédant l'attribut WAI-ARIA: role="img") porteuse d'information a-t-elle une alternative textuelle ?

Test 1.1.2 :

- Chaque zone d'une image réactive (balise <area>) porteuse d'information a-t-elle une alternative textuelle ?

Test 1.1.3 :

- Chaque bouton de type <image> (balise <input> avec l'attribut type="image") a-t-il une alternative textuelle ?

Test 1.1.4 :

- Chaque zone cliquable d'une image réactive côté serveur est-elle doublée d'un mécanisme.

Figure 4 - Exemple de règle du RGAA

Opquast

Qualité Web : la checklist de référence

Rechercher un mot-clé :

Filtrer les règles

Règle n°1 : Les informations sont accessibles à tous les utilisateurs.

Règle n°2 : Les informations sont accessibles à tous les utilisateurs.

Règle n°3 : Les informations sont accessibles à tous les utilisateurs.

Figure 5



Giulia Bianchi

Je suis senior data scientist chez CybelAngel. Je suis passionnée par la data science reproductible et j'ai une approche pragmatique. Je crois en l'importance du partage de connaissances et je contribue activement à la communauté en parlant en conférence et en écrivant des articles techniques. J'ai cofondé et organisé une conférence pendant ses deux premières éditions : DataXDay.

Chercher l'aiguille dans... les données

Lorsqu'on recherche des signaux faibles dans la donnée, l'un des défis auxquels on est confronté lors de l'entraînement d'un modèle de classification est de gérer des classes très déséquilibrées. Dans cet article je parlerai des méthodes pour aborder ce problème et avoir un modèle performant.

Pour introduire la problématique que l'on va traiter dans cet article, je vous propose l'expérience de pensée suivante : imaginez avoir devant vous un bac rempli de boules noires et blanches. Avec les yeux fermés, on vous demande de sortir une boule à la fois et de deviner sa couleur. Étant donné que vous ne voyez pas et que vous n'avez pas d'information additionnelle, il ne vous reste qu'à choisir au hasard entre noir et blanc. Vous avez devant vous un choix binaire avec 50% de probabilité de deviner la bonne couleur pour chaque boule. Si dans le bac la répartition de couleur est moitié-moitié, vous allez vous tromper plus ou moins une fois sur deux.

Et si je vous disais que dans le bac il n'y avait qu'une seule boule noire et que toutes les autres étaient blanches ?! Une stratégie possible consiste à choisir tout le temps la couleur blanche : vous êtes sûrs de vous tromper uniquement une fois ! Très bon résultat du point de vue de l'erreur commise, mais hélas si l'objectif était de retrouver la boule noire vous n'aurez pas eu de succès.

Dans le métier de data scientist, on ne nous demande pas de faire exactement ce genre d'exercice, en revanche on se trouve face à des problématiques qui peuvent y ressembler. La maintenance prédictive, la détection de fraude, un filtre anti-spam ont tous en commun la recherche d'un phénomène rare qui se présente peu de fois, mais qui, lorsqu'il se présente, est embêtant. Selon le cas d'usage "embêtant" peut vouloir dire "coûteux" ou "dangereux" ou "pas agréable" pour l'utilisateur ou les trois à la fois. C'est dans ce contexte qu'un data scientist peut être amené à utiliser des données historiques pour détecter et prédire ce genre de phénomènes rares.

Le but de cet article est de détailler les problématiques qui se présentent dans le cadre de la prédiction d'événements rares et les techniques pour les affronter.

Pour bien pouvoir suivre il faudrait avoir des notions de *machine learning* supervisé : comprendre ce que cela veut dire de faire du *feature engineering*, entraîner un modèle et évaluer ses performances. Je vais aussi citer des méthodes issues de python telles que [pandas](#), [scikit-learn](#) et [xgboost](#) qui sont très utilisées par les *data scientists*.

La classification binaire

En *machine learning*, quand on est face à une tâche qui consiste à distinguer entre 2 catégories (noir et blanc par exemple) et qu'on a des données historiques grâce auxquelles on peut apprendre, on a recours à la classification binaire.

Dans les cas triviaux les deux catégories (ou classes) sont équilibrées. C'est-à-dire qu'on a autant d'occurrences dans une catégorie que dans l'autre. Si c'est le cas, la tâche est alors relativement triviale : après une étape de *feature engineering* nous pouvons entraîner un modèle et évaluer ses performances. Il s'avère qu'avoir des classes équilibrées est un luxe et presque une utopie, car dans la plupart des cas d'usage réels les classes sont déséquilibrées : une des deux catégories, la classe "majoritaire", a beaucoup plus d'occurrences que l'autre, la classe "minoritaire". Et c'est là que cela se complique ! Car un modèle de *machine learning* pour faire une prédiction essaie aussi d'optimiser une quantité (comme dans le tout premier exemple, même s'il n'agit pas avec les yeux fermés) !

Pour mieux saisir la difficulté, prenons le cas de figure d'un jeu de données avec des classes équilibrées :



Ici, la classe positive est constituée de boules noires et la classe négative de boules blanches, les deux ont la même proportion.

Imaginons que le modèle aurait sorti les prédictions :

- Prédiction = "blanche"



- Prédiction = "noire"



Globalement il y a uniquement deux éléments mal classifiés. Pour quantifier l'erreur, on peut utiliser l'[accuracy score](#), une métrique qui consiste à compter le nombre relatif d'éléments mal classifiés. Cela correspond ici à 0,8 (2 erreurs sur 10 éléments).

Maintenant, prenons un jeu de données avec les classes déséquilibrées :



La classe positive (boules noires) représente seulement 10 % du jeu de données et la classe négative le restant (90 %).

On pourrait avoir la même erreur absolue dans le scénario suivant :

Prédiction = "blanche"



Prédiction = "noire"



Le modèle réalise uniquement deux classifications incorrectes et l'*accuracy score* vaut 0,8 également. Dans les deux cas, le modèle performe plutôt bien, car il a bien classifié 8 éléments sur 10, sauf que le deuxième modèle ne sert à rien !

Comment faire ?

Partons de l'hypothèse que l'échantillon de données à disposition pour l'apprentissage est suffisamment grand et représentatif et que le *feature engineering* est bon aussi. Avoir des données pertinentes de qualité et en quantité suffisante et bien travailler sur les *features* restent des incontournables pour bien faire du *machine learning* tout court, peu importe si le jeu de données est équilibré ou pas. Ici l'idée va être de trouver et actionner différents leviers agissant tout au long du processus de modélisation. Voici les axes sur lesquels on peut agir :

- 1 Les données : rééquilibrer les classes
- 2 L'entraînement : modifier le poids des classes
- 3 L'évaluation : utiliser une bonne métrique
- 4 La prédiction : choisir le bon seuil de probabilité

Action sur les données : rééquilibrer les classes

Rééquilibrer les classes consiste à revenir à l'équilibre de proportions entre la classe majoritaire et minoritaire pour que cette dernière soit aussi représentée que l'autre. Il y a deux options : soit on ajoute des occurrences de la classe minoritaire, soit on en enlève à la classe majoritaire.

Comment ajouter des éléments à la classe minoritaire alors qu'on n'en a déjà pas suffisamment ?

La première idée est d'augmenter le nombre limité de données que l'on a à disposition, on parle d'*oversampling*. Une façon basique de faire consisterait à utiliser plusieurs fois tout ou partie des éléments de la classe minoritaire : les dupliquer tout simplement. Le risque évident de surapprentissage (le modèle apprend par cœur sur le jeu de données d'entraînement perdant son pouvoir de généralisation à des nouvelles

données) est assez important, il faut faire attention. Si les performances sur un jeu de données de validation sont trop basses comparées au jeu d'entraînement c'est que le modèle sur-apprend (*overfit*).

Une autre option consiste à générer des éléments synthétiques : pour cela il y a des techniques comme [SMOTE \(Synthetic Minority Over-sampling Technique\)](#) qui permettent de créer des données vraisemblables à partir des vraies données de la classe minoritaire. Le risque ici est de générer des données que concrètement on ne va jamais voir en production. Le modèle aurait de meilleures performances à l'entraînement, mais cela ne sera pas forcément très utile en phase de prédiction en prod. Prenons à nouveau l'exemple du bac à boules. On pourrait s'imaginer rajouter des nouvelles boules noires avec une surface légèrement rugueuse, pas aussi lisse que toutes les autres boules déjà présentes. Vous allez peut-être apprendre que cette surface rugueuse est associée à la couleur noire, mais cela ne vous aidera pas à reconnaître la seule boule noire lisse (l'originale).

La deuxième technique est assez évidente : on choisit au hasard la quantité de données à garder et c'est tout, avec la méthode [sample de pandas](#) par exemple. On parle d'*undersampling*. L'avantage de cette technique est sa simplicité, l'inconvénient est que potentiellement on peut perdre de l'information discriminante sur la classe majoritaire.

On peut choisir l'une de deux techniques, ou combiner les deux, en s'assurant bien de la capacité de généralisation du modèle sur un jeu de données de validation. Pour l'implémentation d'over et under-sampling, il y a une librairie [imbalanced-learn](#) qui fait partie de l'univers des [projets compatibles avec scikit-learn](#).

Vu qu'on parle d'*undersampling*, c'est le bon moment pour rappeler un autre risque lié aux classes très déséquilibrées quand le sous-échantillonnage est effectué intrinsèquement pendant un entraînement de modèle sur le jeu de données entier. Avec les méthodes d'ensemble comme les [random forests](#) et [xgboost](#), ou pendant la [cross-validation](#), un sous-ensemble de données est sélectionné pour chaque itération d'entraînement : il faut alors faire attention à ce que le modèle puisse voir quelques éléments de la classe minoritaire à chaque itération. Dans *scikit-learn*, il y a la méthode [StratifiedKfold](#) pour s'en assurer pendant la *cross-validation*. De la même manière, les paramètres *bootstrap* et *max_features* d'une *random forest* et *subsample* et *sampling_method* de [xgboost](#) permettent de régler le problème au niveau du modèle.

Action sur l'entraînement : modifier le poids des classes

Le levier d'action qui permet d'agir pendant l'entraînement du modèle consiste à considérer les éléments de la classe minoritaire comme étant plus importants. Techniquement, on va modifier l'erreur de classification dans la fonction de coût pendant l'entraînement.

Prenons le cas de la [régression logistique](#) et sa fonction de coût par défaut, une [log-loss](#) :

$$J_{\theta} = -\frac{1}{m} \sum_{i=1}^m Y_i \log P_i + (1 - Y_i) \log(1 - P_i)$$

Où :

m : nombre d'occurrences

Y_i : vraie classe de l'occurrence i

P_i : classe prédite de l'occurrence i

La même fonction de coût avec la correction de poids devient :

$$J_{\theta} = -\frac{1}{m} \sum_{i=1}^m W_0 Y_i \log P_i + W_1 (1 - Y_i) \log (1 - P_i)$$

Où :

W_0 : poids de la classe majoritaire

W_1 : poids de la classe minoritaire

Et $W_1 > W_0$. [Référence: [How to Improve Class Imbalance using Class Weights in Machine Learning](#)]

Cela se fait avec le paramètre "class_weight" des classifieurs de scikit-learn, où :

- class_weight=None (valeur par défaut), on assume que les classes sont équilibrées et il n'y a pas de correction de poids
- class_weight="balanced", le modèle assigne un poids qui est inversement proportionnel aux proportions des classes : $w_j = n_{\text{samples}} / (n_{\text{classes}} * n_{\text{samples}_j})$ for j in n_{classes}

[Référence: [compute class weight](#)]

L'équivalent existe pour [xgboost](#) également : "scale_pos_weight".

Action sur l'évaluation : utiliser une bonne métrique

On a vu que l'*accuracy score* est une métrique d'évaluation peu adaptée aux classes déséquilibrées, car cela ne distingue pas l'erreur commise sur la classe majoritaire ou minoritaire. Voyons alors quelles autres options s'offrent à nous pour pénaliser davantage les erreurs de classification sur la classe minoritaire.

C'est le moment de présenter la [matrice de confusion](#). En *machine learning* supervisé, on a toujours à disposition la prédiction du modèle et la vraie valeur de la variable cible à partir de laquelle le modèle apprend pendant l'entraînement. En cas de régression (la variable cible est une valeur numérique continue) cela permet de calculer des quantités comme la [MSE](#) (Mean Squared Error, [erreur quadratique moyenne](#)). Dans le cas de la classification, après l'entraînement on obtient la classe "prédite" et on peut la comparer avec la "vraie" classe. On distingue alors 4 catégories :

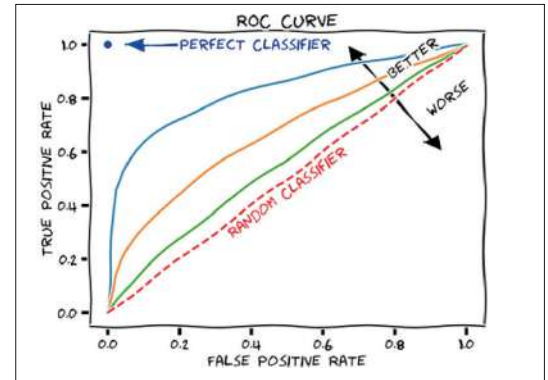
- Les vrais positifs (True Positives - TP) : les éléments de la classe positive correctement classifiés
- Les vrais négatifs (True Negatives - TN) : les éléments de la classe négative correctement classifiés
- Les faux positifs (False Positives - FP) : les éléments de la classe négative qui ont été classifiés comme positifs (erreur de type 1)
- Les faux négatifs (False Negatives - FN) : les éléments de la classe positive qui ont été classifiés comme négatifs (erreur de type 2)

L'*accuracy score* consiste à sommer les vrais positifs et les vrais négatifs et les diviser par le nombre total d'éléments. Il y a d'autres façons d'exploiter la matrice de confusion pour quantifier l'erreur de classification et ainsi la performance du modèle.

Une métrique communément utilisée pour la classification binaire est la [courbe roc](#). La courbe est construite avec deux quantités : *False Positive Rate* sur l'axe des abscisses et *True Positive Rate* sur l'axe des ordonnées.

$$FPR = \frac{FP}{FP + TN} \quad TPR = \frac{TP}{TP + FN}$$

Le classifieur parfait (celui qui ne commet aucune erreur) est celui avec une courbe roc qui passe par le point en haut à gauche du graphe : avec $TPR = 1$ et $FPR = 0$ (aire sous la courbe de 1). Au contraire, une classification basée sur le hasard, présente une courbe roc qui traverse en diagonale (aire sous la courbe de 0.5).

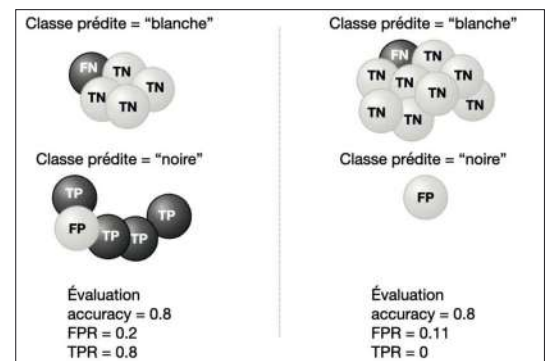


[Référence: [Receiver operating characteristic](#)]

Étant donné que l'objectif est de maximiser l'aire sous la courbe pour tendre au classifieur parfait il faudrait :

- Maximiser les vrais positifs (TP) et les vrais négatifs (TN)
- Minimiser les faux négatifs et positifs (FN et FP)

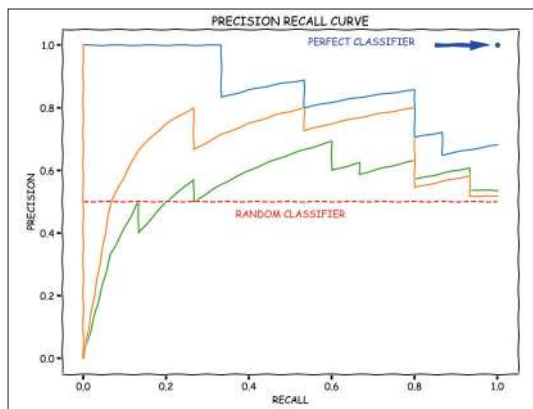
Avec des classes déséquilibrées, les positifs sont peu nombreux, donc les vrais et faux positifs (TP et FP) sont naturellement peu nombreux. Les vrais négatifs (TN) au contraire sont naturellement nombreux. Quand la courbe roc a une valeur qui nous semble acceptable, on peut avoir l'illusion de bonnes performances alors que la capacité de détecter les vrais positifs n'est pas bien captée par cette métrique.



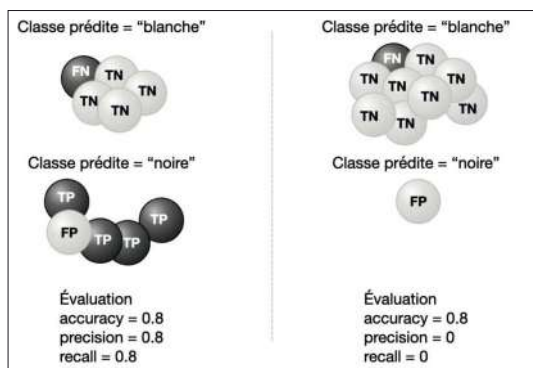
Dans cet exemple, la TPR montre que le modèle n'est pas performant alors que l'*accuracy score* et le FPR sont plutôt bons. Une autre métrique qui peut aider est la [courbe precision-recall](#). Cette courbe est construite avec deux quantités : "Precision" sur l'axe des ordonnées et "Recall" (même indicateur que le True Positive Rate) sur l'axe des abscisses.

$$Precision = \frac{TP}{FP + TP} \quad Recall = \frac{TP}{FN + TP}$$

Le classifieur parfait (celui qui ne commet aucune erreur) est celui avec une courbe precision-recall qui passe par le point en haut à droite du graphe : avec $precision = 1$ et $recall = 0$ (aire sous la courbe de 1). Au contraire, une classification basée sur le hasard, présente une courbe precision-recall qui traverse en horizontale le graphe (aire sous la courbe de 0.5).



L'objectif reste le même que pour la courbe roc, sauf que cette fois les vrais négatifs (TN) ne biaisent plus la perception des performances, car ils n'apparaissent plus dans les formules.



Maintenant, on a deux quantités qui montrent que le modèle n'est pas bon.

Action sur la prédiction : choisir le bon seuil de probabilité

Les modèles pour la classification binaires produisent deux outputs :

- La classe prédite de chaque élément (0 ou 1)
- Un score pour chaque élément, qui varie entre 0 et 1.

Selon le type de modèle, ce score peut correspondre à la probabilité d'un élément d'appartenir à une classe. Généralement on obtient le score pour la classe 0 et le score pour la classe 1, la somme des deux étant égale à 1. Celui qui nous intéresse le plus est le score de la classe positive. Le score et la classe fournis pour un élément sont liés de la façon suivante : par défaut, si le score de la classe positive est supérieur à 0.5, la classe prédite vaut 1, autrement la classe prédite vaut 0. Cela est acceptable quand les classes sont équilibrées, car le score représente la proportion d'occurrences positives dans l'échantillon. Or, vous l'avez désormais compris, avec des classes déséquilibrées ce n'est pas le cas. Comme en fin de compte ce qui nous intéresse est d'avoir

une classe plus qu'un score, ce qu'on peut faire est choisir le seuil à partir duquel trancher de manière plus avisée. Comme point de départ, on peut choisir le seuil qui correspond à la proportion d'occurrences positives dans l'échantillon, mais cela n'est pas toujours le plus approprié.

Il faut remarquer que selon le seuil choisi, toutes les métriques de la matrice de confusion changent, on va alors se servir de ces métriques pour faire un choix. D'un point de vue quantitatif, jongler avec le seuil nous permet d'évaluer la variation de la *precision* et du *recall* par exemple, et on peut choisir le seuil qui maximise l'un ou l'autre ou une métrique pondérée entre les deux comme le *F1 score* (une moyenne harmonique de *precision* et *recall*). En revanche, toute réflexion strictement quantitative n'est pas forcément suffisante pour parvenir au meilleur choix. Ce qui est vraiment intéressant est de faire des considérations de type qualitatif qui dépendent du cas d'usage. Maximiser la *precision* au détriment du *recall*, peut être une bonne solution si l'on veut être sûr de ne rater aucun vrai positif (et les faux positifs ne sont pas un problème). Au contraire, maximiser le *recall* au détriment de la *precision* permet d'être plus conservateur : on va prédire comme positifs un moindre nombre d'éléments, mais parmi eux on est sûrs de trouver plutôt des vrais positifs et peu de faux positifs.

Prenons par exemple le cas d'usage de la maintenance prédictive : une compagnie d'aviation voudrait savoir quelles pièces de ses avions sont à changer juste avant qu'elles ne cassent (et provoquent un accident). Il s'agit de classification binaire, car pour chaque pièce on va se demander si oui ou non elle est à changer et la plupart du temps la réponse est non (classes déséquilibrées). Maximiser la *precision* implique de changer des pièces plus souvent que nécessaire. On est sûr que toutes les pièces sont en bon état, mais cela peut coûter cher, car on répare/change la pièce trop tôt. Maximiser le *recall* signifie que toute pièce changée était effectivement à changer, mais il en reste quelques-unes qui auraient dû être changées aussi... Bref, trouver le bon compromis dépend vraiment du cas d'usage, il n'existe pas de réponse universelle.

Conclusion

Pour résumer, la classification binaire avec classes déséquilibrées présente des challenges additionnels à la classification binaire où les deux classes ont le même nombre d'éléments. Si les données à disposition constituent un échantillon représentatif et que l'étape de *feature engineering* a été menée de façon exhaustive, les options suivantes sont à disposition :

- Intervenir sur le jeu de données pour augmenter la proportion relative de la classe minoritaire avec du *over-* et/ou *under-sampling*
- Augmenter le poids de la classe minoritaire pendant l'entraînement pour que l'erreur sur cette classe soit davantage prise en compte
- Choisir la métrique qui ne donne pas des performances trop optimistes
- Jongler avec le seuil de prédiction pour s'adapter au cas d'usage en interprétant les résultats



Anne-Laure Gaillard

Spécialisée dans la qualité logicielle, passionnée par le test logiciel et l'agilité, partisane de la doctrine « La qualité est l'affaire de tous ».

Tester une API REST sans tomber dans les pommes (d'API)

Les API exposent des données ou des fonctionnalités, la nécessité de les tester n'est plus à démontrer. Cependant si votre stratégie de test d'API consiste à faire quelques appels puis de comparer les résultats [1], je vous propose dans cet article les éléments indispensables pour définir une stratégie de test pour les API REST.

Les API permettent à deux systèmes d'interagir l'un avec l'autre. Elles sont utilisées par les géants du numérique tels que Google, Amazon, Facebook [2], mais aussi par le service public français [3]. De nombreux modèles d'API existent comme SOAP [4], cependant l'architecture REST, est considérée comme plus robuste [5]. La suite de cet article suppose que votre API à tester est une API REST [Figure 1], l'architecture REST repose sur les principes suivants : ressources, verbes et codes HTTP, contrôles hypermédias [6], cependant les concepts présentés dans l'article restent tout de même transposables aux autres types d'API.

je conseille d'apporter une vigilance particulière sur les points suivants.

L'introduction

L'introduction de la documentation doit permettre de comprendre ce que fait l'API, il faut supposer que la personne qui lira ce descriptif n'a aucune idée des systèmes internes, ni du choix d'implémentation. La qualité de la rédaction de ce descriptif est importante, surtout si l'API est publique.

Les ressources

Elles font références aux catégories d'information que l'API peut renvoyer et doivent comporter une description. Sont-elles pertinentes et suffisantes ?

Les points d'accès et les méthodes

Chaque ressource comporte des points d'accès et des méthodes. Un endpoint, que je traduis délibérément par point d'accès, indique comment accéder à la ressource, tandis que les méthodes déterminent les actions, par exemple GET (lecture), POST (création), PUT (mise à jour), DELETE (suppression). Chaque point d'accès doit contenir une description et un chemin, il est recommandé que ce dernier soit constitué de noms et pas de verbes.

Bonne pratique	GET	/users/
Mauvaise pratique	GET	/getallusers/

Les paramètres

Si le point d'accès a des paramètres, les informations fournies pour ceux-ci doivent être suffisantes. Le paramètre doit-il être renseigné dans l'en-tête, dans le chemin de l'URL, en tant qu'argument de requête ? Est-il obligatoire ? Quels sont les formats supportés (chaînes de caractères, entier, booléen, etc.) ? Quelles sont les valeurs minimales et maximales ?

Des exemples de paramètres, de requêtes et de réponses

Les exemples sont incontournables, en plus d'être une excellente pratique de documentation, ils serviront pour les tests, par exemple pour mettre en place des mocks.

Les tests fonctionnels

Ensuite, les tests fonctionnels, généralement ceux qui sont réalisés lorsque que l'on teste quelques appels. De nombreux outils peuvent être utilisés : Postman, SoapUI, Katalon Studio, etc. [9], il ne faut pas hésiter à les comparer afin de



Figure 1 :
Schéma d'une API REST

Les spécifications

Une API est essentiellement un contrat au sein duquel la documentation représente l'accord entre les deux systèmes, un premier test peut alors être réalisé, préalablement à l'implémentation de l'API, ou à défaut avant de lancer les appels sur celle-ci, à savoir la revue de cette documentation. Ceci est un test statique, c'est-à-dire sans exécution du code, au même titre que les revues de code, ces tests sont peu coûteux et permettent de détecter les erreurs au plus tôt, ainsi ils sont considérés comme ayant un bon retour sur investissement [7]. Revenons sur l'API, en plus de vérifier que les besoins vont être pris en compte, cette revue est indispensable pour s'assurer que l'API offre la meilleure expérience possible pour la personne qui aura comme point d'entrée cette documentation. À titre personnel, j'apprécie tout particulièrement la pertinence de celle de GitHub [8].

La documentation comporte plusieurs types d'informations,

trouver celui qui correspond le mieux à son contexte.
 Pour chaque requête d'API il est possible, à minima, de vérifier :

- L'en-tête,
- La réponse,
- Le code d'état HTTP.

Pour ce dernier point, je suppose que vous connaissez le célèbre **404 Not Found**, mais il existe bien d'autres codes [10] **[Figure 2]**, la documentation de l'API doit permettre de connaître le code d'état attendu pour chaque appel.
 Exemple pour tester sur postman le code retour 201 - Created :

```
pm.test("Status code is 201", function () {
    pm.response.to.have.status(201);
});
pm.test("Status code name is Created", function () {
    pm.response.to.have.status("Created");
});
```

Test de chaque requête unitairement

Premièrement, il est plus raisonnable de tester chaque requête de manière isolée en commençant par le *Happy path*, c'est-à-dire le test qui représente l'utilisation idéale, il ne doit pas contenir d'exception, ni générer d'erreur.

Je préconise d'utiliser les paramètres les plus probables, qui peuvent être connus du métier ou déductibles via une analyse de données [11]. Par exemple, si l'API prend en paramètre un nom de famille, le nom le plus fréquent au monde est Wong, cependant si l'API est destinée à la France, le nom de famille le plus probable est Martin.

```
GET /user?userName={userName}
avec userName = « Martin »
```

Pour continuer, on teste les paramètres facultatifs et des valeurs plus complexes.

```
GET /user?userName={userName}&userFirstName={userFirstName}
avec userName = « Prud'homme » et userFirstName = « Claire »
```

Afin d'avoir une couverture plus grande, les tests fonctionnels sont à compléter par des tests négatifs. Que se passe-t-il si les valeurs obligatoires ne sont pas renseignées ? Si elle dépasse la taille maximale autorisée ? Si le format n'est pas correct ? Pour chaque cas négatif un message indique-t-il clairement le problème ?

```
GET /user?userName={userName}
avec userName = « » ou sans userName
```

Test de combinaison de plusieurs requêtes

Chaque requête ayant été testée individuellement, viennent ensuite les tests de scénario avec plusieurs appels qui s'enchaînent :

```
POST /user?userName={userName} création d'un utilisateur
GET /user?userName={userName} l'utilisateur existe
```



```
DEL /user?userName={userName} suppression de l'utilisateur
GET /user?userName={userName} l'utilisateur n'existe pas
```

Figure 2 : Les codes retours des API regroupés en 5 catégories

Il est également possible d'appeler la même requête plusieurs fois de suite :

```
POST /user?userName={userName} création d'un utilisateur
POST /user?userName={userName} impossible l'utilisateur existe déjà
```

Les tests non fonctionnels

Parfois, trop focalisé sur les tests fonctionnels, on en arrive à faire l'impasse sur les tests non fonctionnels qui sont pourtant essentiels à la pérennité des API, notamment les tests liés à la sécurité et la performance de l'API.

La sécurité

Les API gèrent des données potentiellement sensibles et génèrent du chiffre d'affaire, ainsi la sécurité de l'API est essentielle [12] pourtant de nombreuses vulnérabilités existent, parmi le top 10 des failles de sécurité d'API on retrouve l'accès aux objets, voire aux ressources sans autorisation [13]. Pour les API nécessitant une authentification et des autorisations, les tests de base peuvent être :

- désactiver l'authentification,
- renseigner un mot de passe ou jeton erroné,
- utiliser un jeton périmé pour vérifier que les accès sont bloqués,
- s'authentifier avec un compte n'ayant pas l'autorisation de réaliser une action.

Les codes retours dans ces cas sont généralement 401 Unauthorized et 403 Forbidden.

Enfin, le protocole doit être conforme à l'attendu : http/https.

Les performances

Les performances sont importantes (à critiques) selon les projets. Quel est le temps de réponse, la latence ? Dans un premier temps, une simple vérification du temps de réponse des tests précédents ne devrait pas être compliquée.

Exemple pour ajouter sur postman un test sur le temps de réponse :

```
pm.test("Response time is less than 1000ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(1000);
});
```

Pour aller plus loin, il est possible d'ajouter des variables pour les appels afin de contourner le cache et de créer plusieurs scénarios avec lecture/écriture. Comment l'API gère-t-elle la charge ? Quelle est la consommation du CPU ? Combien de requêtes sont gérées par seconde ? Comment est gérée la panne ? Ces tests peuvent être partiellement effectués via vos

outils de tests d'API classiques, pour des tests plus avancés, il faudra utiliser des outils dédiés tels que JMeter, HPE LoadRunner, ApacheBench ou Siege.

Et après ?

Je vous conseille d'automatiser vos tests et de les intégrer à vos outils de CI/CD habituels [14]. Ainsi les tests seront joués à chaque modification, vous détecterez au plus tôt des éventuelles régressions et vous gagnerez en confiance sur la qualité de vos livrables.

Vous pouvez également enrichir les tests avec des données aléatoires, notamment pour faire du fuzzing [15], en mode data-driven testing ou encore combiner avec des tests IHM. Pensez dans tous les cas à échanger avec votre équipe et sur les communautés que vous fréquentez, les développeur·e·s, testeur·e·s, expert·e·s données ou métiers auront certainement des suggestions pertinentes ! Vous pouvez d'ailleurs les solliciter pour réaliser des tests exploratoires [16].

Votre API est en production, félicitations ! Avez-vous envisagé un monitoring ? Ce service de surveillance vous enverra une alerte pour vous informer de l'état de santé de votre API en vérifiant, la vitesse, le contenu et les codes de réponse de l'API.

En conclusion, les indispensables des tests d'API sont la pratique des revues et l'automatisation des tests fonctionnels et non fonctionnels, en variant les données. Pour aller plus loin et faire grandir votre stratégie pour les tests d'API, à vous d'innover et de solliciter votre équipe afin que celle-ci soit enrichie et partagée [Figure 3].

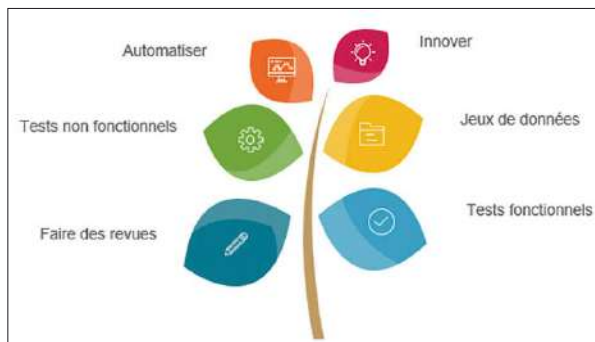


Figure 3 : Les concepts principaux des tests d'API

- [1] <https://www.programmez.com/avis-experts/trois-raisons-de-tester-soigneusement-les-apis-24364>
- [2] <https://www.journauldunet.com/solutions/dsi/1197015-les-api-pourquoi-elles-sont-devenues-incontournables-pour-l-innovation/>
- [3] <https://api.gouv.fr/rechercher-api>
- [4] <https://apievanglist.com/2018/02/03/api-is-not-just-rest/>
- [5] <https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest>
- [6] <https://martinfowler.com/articles/richardsonMaturityModel.html>
- [7] <https://latavernedutesteur.fr/2017/11/03/les-tests-statiques-rois-du-roi/>
- [8] <https://docs.github.com/en/rest>
- [9] <https://sourceforge.net/software/api-testing/>
- [10] <https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10>
- [11] Moustier, C., **Le test en mode Agile**, 2019, éditeur ENI, collection Datapro, 657p, chapitre 4 Versant industriel du test, section Optimisation des tests par les valeurs.
- [12] <https://www.programmez.com/avis-experts/la-securite-des-api-un-defi-grandissant-pour-les-entreprises-29027>
- [13] <https://owasp.org/www-project-api-security/>
- [14] Gregory, J et Crispin, L, Ayadi, E, Butel, B, **Agile Testing Condensed, Version française : Une brève introduction**, 2021, 111p, chapitre 8 Tester dans un Contexte DevOps et chapitre 10 Visualisation d'une Stratégie d'Automatisation des Tests.
- [15] <https://owasp.org/www-community/Fuzzing>
- [16] Hendrickson, E., **Explore It! Reduce Risk and Increase Confidence with Exploratory Testing**, 2013, Carter, J., Pragmatic Bookshelf, 162p, section Exploring an API.

LE NUMÉRIQUE C'EST ÉCOLOGIQUE SI...



Viktoriia Melnyk

J'ai 17 ans, je suis ukrainienne et ça fait 5 ans que j'habite en France. Je suis actuellement en Terminale au lycée

Jacques Monod à Clamart. Je suis très passionnée par les sciences et le numérique, récemment j'ai pu créer un site sur le développement personnel.

La création des ordinateurs nous a fait économiser beaucoup de papier et nous avons l'impression que le numérique a un impact positif sur l'écologie mais bien souvent vous vous trompez puisque toutes les données prennent beaucoup de place dans les datacenters. Imaginez quels changements peuvent se produire si chacun apporte un peu d'aide à l'écologie. Quelques exemples très simples : vider votre boîte mail, 80 % de messages ne sont jamais ouverts, utiliser un moteur de recherche qui présente un programme écologique, reforestation par exemple. De plus le streaming est très polluant, téléchargez vos musiques !

Ce poème est adressé à notre planète qu'il faut continuer d'aimer et faire beaucoup attention à elle !

Des petits plaisirs de vie

Notre Terre est Belle, elle a des milliards d'années
Elle a tout prévu pour nous mettre le plus à l'aise possible
Manger, marcher et respirer
Que veux-tu de plus pour profiter ?
Elle était sage et libre comme un oiseau qui vole au-dessus d'un océan puissant et tellement profond
Elle a ses plans et ses occupations
Jusqu'à ce que tu l'as piège et coupes ses ailes
C'est des blessures à vie
Elle en peut plus, elle crie, te prie depuis longtemps
Ne sois pas indifférent et arrête-toi pour un seul instant
Tu penses que cela ne va jamais te concerner, avec le temps tout va s'oublier
Mais tu as tort, ses blessures commencent déjà à se montrer
Pourquoi tu l'as détruit ?
Pour satisfaire tes petits plaisirs absurdes ?
Tu es sûrement intelligent, c'est surprenant
Développer, innover, enfin évoluer
Tu es puissant, l'envie de dominer le monde t'as submergé et aveuglé
La chose est telle que tu n'as pas compris l'essentiel de notre existence toute entière
Dans la forêt tu marches et tu entends les chants de tes oiseaux, l'odeur des fleurs et d'arbres
Les animaux libre, les glaciers solides et surtout ton éclaircissement
Un monde où règne enfin l'amour et réconciliation
C'est ça un vrai plaisir de vivre !

Maîtriser les interfaces en Go

L'une des forces de Go est sa simplicité. Cependant, comme n'importe quel langage de programmation, Go réserve aussi sa part de subtilités que l'on saisit avec l'expérience. Pour ma part, l'adage "accepter des interfaces et retourner des types concrets" [1] a été pendant quelque temps un mystère avant de devenir une évidence. Pour vous faire gagner un peu de temps et vous éviter de reproduire les mêmes erreurs, laissez-moi vous accompagner à travers ce concept...

Le concept

Tout d'abord, que signifie "accepter des interfaces, retourner des types concrets" ? C'est tout simple : lors de la déclaration d'une fonction ou méthode, il suffit de prendre en paramètre des interfaces et d'utiliser des types concrets (généralement struct ou pointeur) en type de retour. Pour comprendre, prenons l'exemple d'un type gérant les articles d'un blog. Ce service pourra sauvegarder des informations dans sa base de données. **Figure 1**

On déclare tout d'abord le type `BlogPost`. Celui-ci contient une dépendance ("store") interagissant avec la base de données.

```
type BlogPost struct {  
    store Store  
}
```

Pour faciliter la création de ce service, on définit la fonction `NewBlogPost`. Elle initialise le service avec la dépendance passée en paramètre. `Store` est ici une interface.

```
func NewBlogPost(s Store) *BlogPost {  
    return &BlogPost{  
        store: s,  
    }  
}
```

Accepter des interfaces

Commençons par le début. Pourquoi accepter des interfaces ?

Flexibilité

Bien entendu, il s'agit ici de rendre notre code **flexible** grâce au principe d'injection de dépendances (le "D" de SOLID [4]). L'interface "Store" nous permet facilement d'interagir avec différentes bases de données (PostgreSQL, MySQL, fichiers, etc.). Cette interface simplifie l'écriture des tests de notre service : Avoir une interface en paramètre permet d'utiliser des mocks. L'avantage ? Pas besoin de requêter la base de données dans les tests unitaires, permettant ainsi de : simplifier le code, tester des situations précises et de réduire leur durée d'exécution.

Compréhension du code

Au-delà de la flexibilité, l'interface améliore aussi la compréhension du code. En restreignant l'interface passée en paramètre aux uniques fonctions dont le code a besoin, on en apprend d'ores et déjà plus sur le comportement du code. C'est le principe de séparation des interfaces (le "I" de SOLID). Si nous utilisons l'exemple du Store interagissant avec la base de données, celui-ci contient notamment les fonctions suivantes :

- `GetBlogPost` qui récupère un article dans la base de données ;
- `CreateBlogPost` qui insère un article en base de données ;
- `CreateUser` permettant de créer le compte d'un utilisateur.

En utilisant directement `Store` en paramètre de la fonction initialisant le service d'article de blog, je risque d'avoir trop d'actions à ma portée. Est-ce que le service gérant les articles de blog peut créer un utilisateur ? Si oui, à quel moment ? Lors de la création du premier article ? Voilà des questions que nous pouvons facilement éviter !

Définissons une interface `BlogPostStore` contenant uniquement les signatures des méthodes dont nous avons besoin et passons-la en paramètre de `NewBlogPost`.

```
type BlogPostStore interface {  
    GetBlogPost(id string) (post *BlogPost, err error)  
    CreateBlogPost(post *BlogPost) error  
}
```

```
func NewBlogPost(s BlogPostStore) *BlogPost
```

Dans cet exemple, on comprend exactement à quoi le paramètre servira : interagir uniquement avec les articles de blog. Le code devient plus lisible pour les prochains développeurs travaillant sur ce projet : `BlogPost` n'a pas pour objectif de modifier les utilisateurs de l'application, mais simplement de gérer les articles de blog.

Retourner des types concrets

Très bien, il paraît clair que prendre des interfaces en paramètre est une bonne idée. Mais pourquoi pas en type de retour ? Pourquoi faudrait-il retourner des types concrets ?

Maintenance et rétrocompatibilité

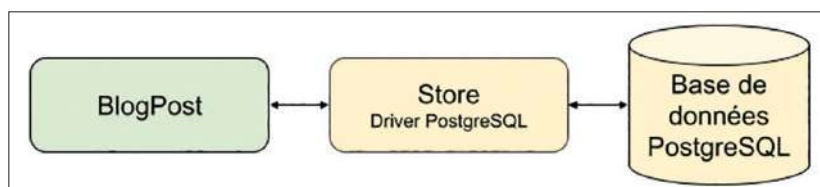
En Go, les interfaces sont définies de manières **implicites**. C'est-à-dire que n'importe quel type qui implémente les méthodes d'une interface implémente l'interface elle-même. Il s'agit de *structural typing* [5] (similaire au *duck typing* pour des langages dynamiques). Il n'est pas nécessaire que le type déclare officiellement qu'il satisfait l'interface. Pour un développeur qui vient d'un langage où les interfaces sont définies de manière **explicites** (C#, Java...), il est plutôt courant de retourner des interfaces. En Go, retourner une interface c'est perdre de l'information.



Lucille Tachet

Ingénieure informatique développant avec Go depuis 5 ans. Elle est Lead Engineer à Lightspeed en Nouvelle-Zélande où elle y co-organise les meetups GoBridge à Auckland. Le but de cette association est d'apprendre le Go aux débutants et plus particulièrement aux groupes sous représentés dans le monde de la tech.

Figure 1. Design du code de gestion d'articles de blog



Par exemple, prenons l'interface suivante que BlogPost implémente :

```
type BlogPostInterface interface {
    GetTitle() (string, error)
    GetContent() (string, error)
}
```

Si on ajoute le champs AuthorID (l'identifiant de l'auteur de l'article) à BlogPost:

```
type BlogPost struct {
    store Store
    AuthorID string
}
```

Retourner une interface ne me permet plus d'y accéder:

```
func NewBlogPost(s store) BlogPostInterface {
    return &BlogPost{store: s}
}

// Création d'un blog post
// blogPost := NewBlogPost(s)
// impossible d'accéder au champ blogPost.AuthorID
```

BlogPostInterface nous permet d'accéder uniquement à GetTitle et GetContent. Le nom de la fonction nous laisse pourtant penser que nous aurons accès à l'intégralité du type: "NewBlogPost". Ainsi, retourner une interface, c'est perdre l'accès à des champs et méthodes exportés. Dommage lorsque l'on sait que cela n'est pas nécessaire !

Astuce : pour faire apparaître explicitement qu'un type doit implémenter une interface, il est possible d'utiliser un identifiant vide [6] par exemple :

```
var _ locator = locationService{}
Cela lèvera une erreur de compilation si l'interface n'est pas satisfaite.
```

Grâce à ses interfaces implicites, Go nous permet d'écrire du code simple : pas besoin de réfléchir à l'avance de quelles interfaces on aura besoin. On peut utiliser un type concret et le remplacer par une interface si nécessaire par la suite. Le code sera toujours compatible ce qui permet d'éviter de compliquer inutilement son code à l'avance.

Retourner une interface en Go, c'est imposer un comportement

Reprenons notre exemple du Store. Cette fois, nous retournons une interface pour comprendre les impacts.

```
type Store interface {
    GetBlogPost(id string) (post *BlogPost, err error)
    CreateBlogPost(*BlogPost) error
    CreateUser(*User) error
}

func NewStoreBad() Store {
    // init db client
    return &Postgres{client: client}
}
```

Le premier effet que l'on peut observer, c'est la tendance avec laquelle on va vouloir réutiliser cette même interface en paramètre d'une autre fonction. Puisqu'on a déjà une interface, pourquoi en définir une nouvelle ? Voyons ce que cela

donne avec notre service de gestion d'articles de blog utilisant l'interface Store.

```
func NewBlogPost(s Store) *BlogPost {
    return &BlogPost{store: s}
}

func main() {
    s := NewStoreBad()
    bp := NewBlogPost(s)
    // ...
}
```

En utilisant Store en paramètre de NewBlogPost, nous commençons plusieurs erreurs:

Tout d'abord, on va ici à l'encontre du principe de séparation des interfaces. Comme expliqué plus tôt : le service d'articles n'a pas besoin de toutes les actions que le Store propose. Ensuite, nous rendons le code moins maintenable. On peut généralement observer un pattern où des méthodes sont ajoutées à l'interface quand le service grandit. Imaginons que j'ajoute une nouvelle fonctionnalité DeleteUser au Store qui permet de supprimer un utilisateur.

```
type Store interface {
    GetBlogPost(id string) (post *BlogPost, err error)
    CreateBlogPost(*BlogPost) error
    CreateUser(*User) error
    DeleteUser(userID string) error
}
```

Mince, je viens de casser mon code ! Tous les types implémentant Store ne sont désormais plus compatibles !

Un cas classique qui permet d'observer ce phénomène concerne les mocks. Ils sont généralement les premières victimes et doivent être modifiés dès qu'une fonctionnalité est ajoutée. Avant l'ajout de la fonctionnalité DeleteUser, voilà à quoi mes tests pouvaient ressembler :

Code complet sur [programmez.com](https://www.programmez.com) & [github](https://github.com)

En ajoutant la fonction DeleteUser au service de localisation et son interface, je suis forcée de mettre le mock à jour en définissant la méthode DeleteUser que je n'utilise même pas dans le service d'articles ! Oups !

```
func (s *storeMock) DeleteUser(userID string) error {
    return nil
}
```

Conclusion

Les interfaces en Go sont des outils puissants qu'il faut savoir maîtriser. Utilisées en paramètre, elles permettent de rendre le code flexible et lisible si elles sont restreintes aux méthodes nécessaires. En revanche, parce qu'elles sont définies de manières implicites, les interfaces peuvent nuire au code si elles sont utilisées comme type de retour. Pour résumer : accepter des interfaces, retourner des types concrets !

Sources:

- [1] <https://github.com/golang/go/wiki/CodeReviewComments#interfaces>
- [2] <https://golang.org/>
- [3] <https://tour.golang.org>
- [4] [https://fr.wikipedia.org/wiki/SOLID_\(informatique\)](https://fr.wikipedia.org/wiki/SOLID_(informatique))
- [5] https://fr.wikipedia.org/wiki/Syst%C3%A8me_structural_de_types
- [6] https://golang.org/doc/effective_go#blank_implements

Un bot #(quasi)nocode sur la Google Home

Depuis l'année dernière, Google¹ a sorti une nouvelle boîte à outils, Google Actions & Builder SDK, pour créer rapidement et efficacement un bot sur les appareils Google dans la mouvance du nocode qui émerge depuis quelques années. L'article va vous permettre d'aborder les premières étapes de création d'un voicebot pour lister les tâches du jour avec Todoist, en français !

Avertissement : cet article n'est pas sponsorisé par Google et Todoist ;).

(1) L'article de blog : <https://developers.googleblog.com/2020/06/announcing-actions-builder-actions-sdk.html>

Le projet (Todoist) et la création du projet sur Google Actions

On va créer un voicebot qui interagit avec votre compte Todoist. Todoist² est un gestionnaire de tâches, sur toutes les plateformes comme Google Home... mais en anglais seulement. Un peu dommage quand on parle français ! Mais heureusement, ils mettent à disposition une API pour communiquer avec leur service. On pourra ainsi communiquer avec notre bot en français pour connaître les tâches à faire dans la journée !

Tout d'abord, on doit :

- Créer un compte sur Google Actions et créer une Action (le petit nom d'une commande vocale pour la Google Home) : <https://console.actions.google.com/>
- Créer une application sur Todoist pour avoir accès à l'API de Todoist : <https://developer.todoist.com/appconsole.html> (compte Todoist requis).

Les API Todoist et l'authentification

Google Home doit pouvoir communiquer avec l'API de Todoist et Google Actions. Il va créer automatiquement tout ce qu'il faut dans le dialogue pour permettre à l'utilisateur de rentrer ses identifiants. L'idée est que le compte Google de l'utilisateur ait les droits d'interroger l'API Todoist.

Pour cela, il faut activer l'*account linking* (dans le menu de la console Google Actions) et rentrer les paramètres suivants, comme le présente l'image³. **Figure 1**

Voice UX : Intentions, Entités, Dialogues

Une interface conversationnelle a besoin de comprendre le message de l'utilisateur (et non à l'utilisateur de savoir où cliquer) et d'y répondre efficacement pour que l'utilisateur ait la meilleure expérience possible.

Pour cela on parle de la notion d'*intention* (intention) qui est déterminée dans la phrase de l'utilisateur. On peut aussi ajouter des paramètres d'intentions, lesquels servent à donner des informations pertinentes supplémentaires.

On va créer un schéma de dialogue pour commencer avec

Figure 1
Interface avec account linking

Google Actions. Le schéma de dialogue est le suivant :

- Ok Google, je voudrais parler à Todoist
- Réponse Google Home (je suis déjà authentifié avec mon compte Todoist)
- Liste-moi les tâches du jour

Il faut tout d'abord créer une liste d'exemples qui va servir à la Google Home à reconnaître l'intention. Il est recommandé d'avoir 10 phrases d'exemple minimum par intention. Les libellés surlignés en bleu correspondent à un paramètre d'intention de type date. **Figure 2**

(2) <https://todoist.com/>

(3) Plus d'informations sur la documentation : <https://developer.todoist.com/guides/#authorization>



Aurore de Amaral

développeuse back-end depuis 2015 et intéressée par toutes les nouvelles technologies autour de l'interface conversationnelle et le traitement du langage naturel (NLP)

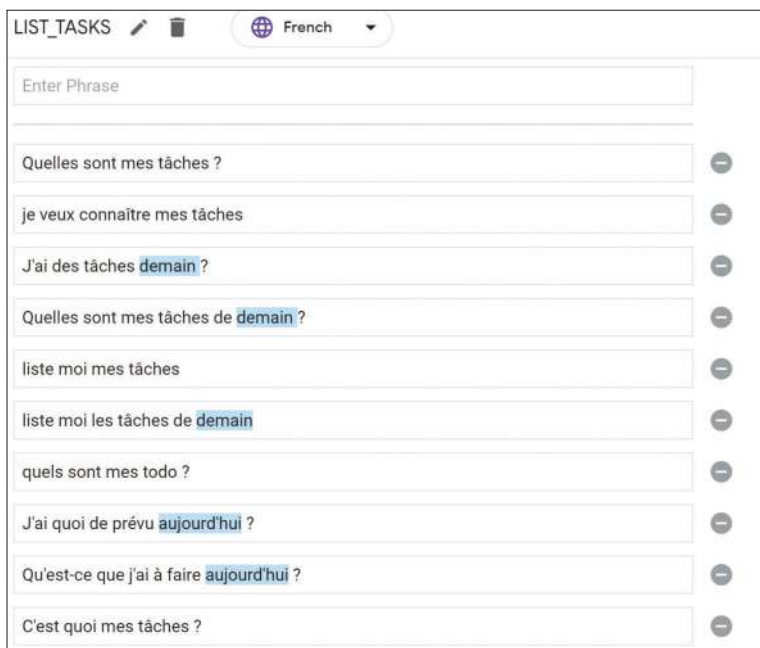


Figure 2 - Interface de définition d'intentions

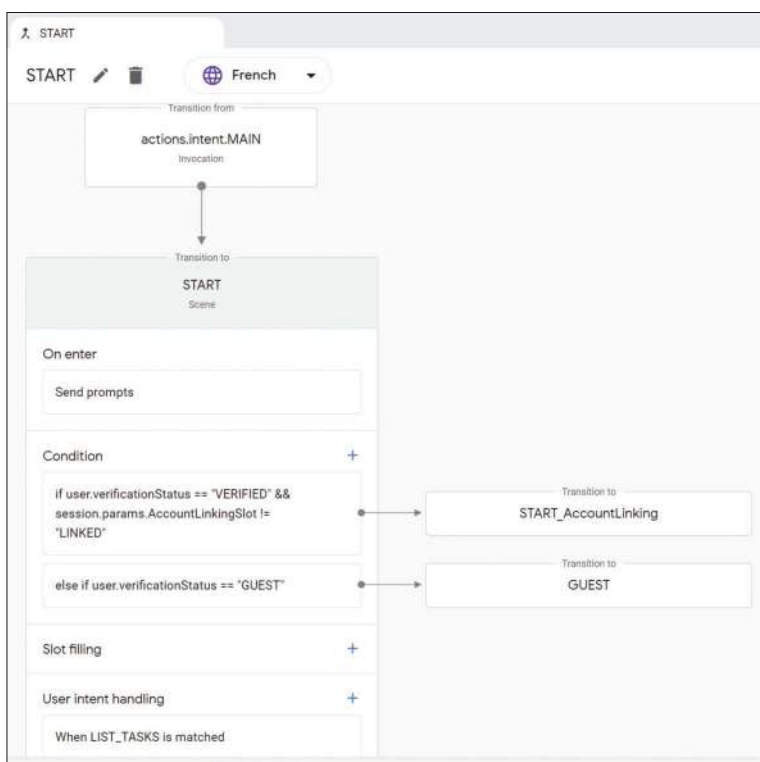
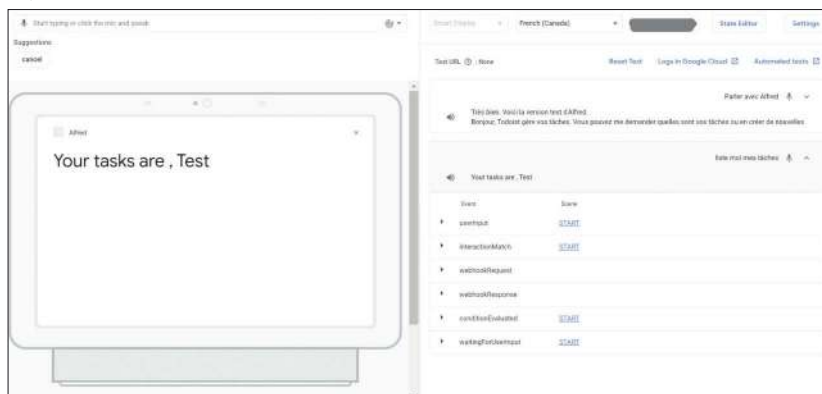


Figure 3 - La scène START

Figure 4



Cette partie utilise DialogFlow⁴ en interne, qui utilise des outils de machine learning (traitement du langage naturel) pour créer un modèle unique reconnaissant nos intentions. Les intentions sont utilisées dans des **scenes**⁵ (scènes) où l'on va déclarer ce que le bot va déclencher après la reconnaissance d'une intention. La conversation étant simple, il suffit de créer une seule scène, GUEST, qui génère l'account linking défini en amont et le code qui sera utilisé quand le bot comprendra l'intention LIST_TASKS. Enfin la redirection vers une autre scène dans le cas où l'utilisateur ne peut ou ne veut pas être authentifié vers l'API Todoist. **Figure 3**

Quand même un peu de code pour l'API Todoist

On ne peut malheureusement pas se passer de code pour interroger l'API de Todoist. L'exemple de départ consistait à lister ses tâches d'aujourd'hui. Pour cela, on peut passer par notre éditeur de code préféré pour le faire, ou via l'interface de Google Actions (moins flexible pour les développeurs). Il faut activer le Webhook (dans le menu de la console Google Actions) et choisir **Inline Cloud Functions**, qui va rediriger vers un éditeur de code, avec un exemple basique. C'est ici qu'il faut rajouter le code métier et l'appel à l'API Todoist⁶. Voici un exemple simple :

```
const { conversation } = require('@assistant/conversation');
const functions = require('firebase-functions');
const todoist = require('todoist-rest-api').default;

const app = conversation();

app.handle('<Your Intent handler Name Here>', conv => {
  functions.logger.info("Received a list_tasks intent");
  const bearerToken = conv.user.params.bearerToken;
  const api = todoist(bearerToken);
  return api.v1.task.findAll({ filter: "aujourd'hui"}).then((tasks) => {
    const voiceTasks = tasks.reduce((acc, task) => { return acc + ' ' + task
    .content; }, "");
    return conv.add(`Your tasks are ${voiceTasks}`);
  }).catch(() => { return conv.add("DEBUG ERROR"); });
});

exports.ActionsOnGoogleFulfillment = functions.https.onRequest(app);
```

Enfin, on doit juste rajouter l'appel au webhook dans la scène START, et le faire à chaque fois que l'on voudra rajouter une intention que doit gérer le webhook. Dans l'onglet test, on peut tester le bot en direct sans passer par une Google Home / Google Nest ou bien sur le téléphone associé à un compte Google. **Figure 4**

Conclusion

Avec un aperçu des fonctionnalités à disposition pour le développeur d'aujourd'hui, la création de bots (chatbot, voicebot,...) est de plus en plus "UX friendly", ce qui tend à une démocratisation du sujet, tout en restant à portée du développeur, qui s'approprie aussi le domaine. Cela permet de faire évoluer peu à peu le métier de développeur, voire même de faire émerger de nouveaux profils !

(4) <https://dialogflow.cloud.google.com/>

(5) Plus d'information dans la documentation : <https://developers.google.com/assistant/conversational/scenes>

(6) Avec l'utilisation d'un paquet npm : <https://github.com/moranej/todoist-rest-api>

Comment configurer PostgreSQL pour la production quand on est débutant.e ?

Ça y est, votre pire peur est arrivée : on vous a demandé d'installer PostgreSQL en production, mais vous n'y connaissez rien et vous êtes un peu perdu.e et ne savez quoi faire de tous les paramètres possibles. Heureusement, voici un petit guide des meilleures pratiques de Postgres en production qui devrait vous aider.

Linux ou Windows

N'espérez pas obtenir le meilleur de Postgres si vous le faites tourner sous Windows. Oui, ça tourne, mais Postgres a été écrit pour Linux, la version Windows n'est qu'un portage, quelle que soit sa qualité.

De plus, sous Windows, vous n'aurez accès qu'à un choix restreint d'outils annexes comme un bon outil de sauvegarde (l'outil proposé par Postgres n'est pas suffisant), un bon outil de haute disponibilité ou l'ajout d'extensions dont votre système pourrait avoir besoin.

Enfin, la majorité des systèmes de production critiques sont sous Linux, vous risquez donc un peu plus de tomber sur un bug inconnu et vous aurez aussi plus de mal à faire corriger ce bug, les bénévoles de la communauté travaillant en majorité sous Linux.

Stockage

Bien entendu, des disques SSD seront plus rapides que des disques mécaniques. Enfin, de *bons* disques SSD seront plus rapides que des disques mécaniques. Il est donc conseillé d'utiliser ce genre de stockage sur un système qui devra absorber une forte charge d'écritures.

Pour paralléliser les pics d'écritures, utiliser un disque différent pour Postgres et pour l'OS est une bonne idée, de même que déporter les journaux WALs sur un troisième disque (flag `-X` ou `--waldir` de l'outil `initdb`).

Au niveau du formatage, le plus recommandé de nos jours est xfs qui permet d'obtenir de bonnes performances tout en restant stable et relativement constant sur les temps de réponse, mais attention, ce format présente des détériorations de performances importantes sur les très grosses tables (>1 To). Cependant, si vous atteignez ce niveau, le partitionnement est fortement conseillé.

Vous pouvez aussi optimiser les I/O en augmentant la valeur de `read_ahead_kb`, ce qui permet de réduire le nombre de requêtes au disque pour le même volume de données lues/écrites. La valeur par défaut est `128kB`, on conseille de mettre plutôt `4096kB` de nos jours.

Dernier paramètre pour le stockage : le scheduler. En effet, la valeur par défaut n'est pas prévue pour une base de données. Il est conseillé de le changer pour `deadline` (CentOS/Red Hat7) ou `mq-deadline` (CentOS/Red Hat 8).

Initialisation de Postgres

Certains paramètres sont difficilement ajustables une fois le cluster PostgreSQL créé. C'est le cas du paramètre gérant la localisation des WAL, la collation par défaut des bases de données, le répertoire des données, l'ajout de checksums sur les pages de données, l'authentification en SRACM-SHA-256 et la non-utilisation de locales spécifiques.

Pour cette raison, il est recommandé d'initialiser PostgreSQL avec les paramètres suivants :

```
initdb \  
--auth-host=scram-sha-256 \  
--pgdata=<pgdata_partition>/data \  
--waldir=<pgwaldata_partition>/wal \  
--encoding=UTF-8 \  
--data-checksums \  
--no-locale
```

Vous devrez aussi mettre à jour le fichier de service de systemd pour qu'il connaisse le bon répertoire PGDATA. Vous trouverez ce service sous `/usr/lib/systemd/system/postgresql-<version>.service`.

Paramétrage de Postgres Configuration et authentification

max_connections : C'est le nombre de connexions simultanées acceptées par Postgres. Il est recommandé de ne pas mettre une valeur trop élevée pour un serveur OLAP (entre 50 et 100). Pour de l'OLTP, on peut monter un peu plus (de l'autre de 4 * le nombre de cœurs CPU. Si ce nombre est inférieur à 100, il est conseillé de mettre 100. Si vous avez besoin de plus de connexions simultanées, pensez à utiliser un pool de connexions comme `pgbouncer`, par exemple.

password_encryption : Pour des raisons de sécurité, il est préférable d'encoder les mots de passe en SCRAM-SHA-256.

Utilisation des ressources

shared_buffers : C'est un des paramètres les plus importants. Il permet de définir combien de mémoire PostgreSQL peut utiliser pour travailler sur les données. Il est difficile de définir la meilleure valeur sans benchmark, car certaines applications fonctionnent mieux avec une petite valeur et pour d'autres il faudra mieux une grande valeur. La recommandation générale est de valoriser ce paramètre à 25% de la RAM.



Laetitia Avrot

Laetitia est Senior Database Consultant pour EDB.

Elle a commencé son aventure Postgres en 2007. Elle a rapidement dû apprendre aussi le métier de DBA Oracle et SQL Server. Elle est impliquée dans la communauté en tant que trésorière de PostgreSQL Europe, cofondatrice de Postgres Women et contributrice reconnue du projet PostgreSQL.

work_mem: C'est le montant maximum de mémoire que chaque opération de tri ou de hash peut utiliser. On peut commencer avec une valeur comme

$(\text{RAM totale} - \text{shared_buffers}) / (16 * \text{nombre de cœurs})$. Il est recommandé de surveiller la création des fichiers temporaires (voir **log_temp_files**).

maintenance_work_mem: C'est la quantité de mémoire que les opérations de maintenance comme les vacuums et les réindexations peuvent utiliser. La valeur par défaut est un peu basse. On peut utiliser comme point de départ

$15\% * (\text{RAM totale} - \text{shared_buffers}) / (\text{nombre de processus autovacuum})$ avec un maximum aux alentours d'1Go. Une surveillance des opérations d'autovacuum permettra de s'assurer que c'est suffisant.

Journaux WAL (Write Ahead Logs)

wal_compression: Il peut être intéressant d'activer la compression des WAL, ça prend un peu plus de CPU, mais cela peut augmenter les performances sur des systèmes créant beaucoup de WAL. Pensez à bencher avant de modifier ce paramètre.

wal_buffers: c'est la mémoire que PostgreSQL peut utiliser pour stocker les WAL. Chaque WAL fait 16Mo, une valeur de 64Mo permet donc de stocker 4 WAL, ce qui devrait être suffisant.

checkpoint_timeout: Il est recommandé de mettre une valeur entre 5 et 15 minutes.

checkpoint_completion_target: Ce paramètre permet de déterminer le temps que prendra le checkpoint. Pour éviter le pic d'I/O, il est recommandé que ce temps soit relativement long. Une valeur de 0,9 (c'est-à-dire 90% de la durée entre deux checkpoints).

archive_mode: Si vous tenez à vos données, l'archivage doit être activé.

archive_command: Il est recommandé d'utiliser un outil de backup qui gèrera l'archivage pour vous (ainsi que la rétention des backups et des WAL correspondant aux backups). Les plus utilisés dans la communauté sont pgBackRest et Barman.

Tuning des requêtes

random_page_cost: Cette valeur doit être ajustée à 1.1 pour les disques SSD. Pour des disques mécaniques, il faudra faire des benchmarks pour déterminer la meilleure valeur pour vos disques.

effective_cache_size: Cette valeur est une approximation permettant à l'optimiseur de savoir si la donnée qu'il cherche a des chances d'être en cache. Une bonne valeur est la somme de **shared_buffers** et du cache de Linux (visible en tapant la commande `free`).

cpu_tuple_cost: C'est le coût processeur pour calculer chaque ligne d'une table résultat. La valeur par défaut est un peu trop prudente. Il est possible d'augmenter à 0,03.

Logs

Pour comprendre ce qu'il se passe sur votre cluster PostgreSQL, vous devez avec des logs les plus verbeux possibles.

log_destination: Je recommande d'utiliser **csvlog**. Ce format

permet, selon moi, les meilleures analyses de logs.

logging_collector: Si vous avez choisi **csvlog**, il faut impérativement mettre ce paramètre sur **on**, sinon, vous n'aurez pas de log.

log_directory: permet de rediriger les logs dans le bon répertoire.

log_checkpoints: Vous voulez pouvoir voir chaque fois qu'un checkpoint a lieu. Il faut mettre ce paramètre sur **on**.

log_lock_waits: Vous voulez voir lorsqu'une requête est bloquée par un verrou, il faut mettre ce paramètre sur **on**.

log_statement: Je considère que logger toutes les requêtes de type **ddl** est une très bonne habitude.

log_temp_files: Vous voulez savoir quand Postgres est obligé de swapper, mettre **0** pour ce paramètre permet de voir tous les fichiers temporaires créés.

log_autovacuum_min_duration: Pour surveiller efficacement les processus d'autovacuum, je recommande de mettre **0** pour voir tous les lancements d'autovacuum.

log_min_duration_statement: Pour un serveur OLTP, **250ms** est une bonne valeur. Pour un serveur OLAP, **1s** est pas mal. Si ce paramètre fait que les logs sont très volumineux, vous pouvez augmenter temporairement la valeur, régler les soucis de performance et redescendre la valeur du paramètre une fois que la situation est normalisée.

Autovacuum

Un cluster PostgreSQL normal nécessite peu de maintenance. C'est en partie grâce au démon d'autovacuum qui va mettre en place cette maintenance automatiquement quand elle est nécessaire. Les paramètres par défaut sont trop conservateurs, ce qui rend cette maintenance peu efficace. Boostons un peu l'autovacuum.

autovacuum_max_workers: la valeur par défaut (**3**) est assez faible, d'autant plus qu'il faut redémarrer le serveur PostgreSQL si on souhaite modifier cette valeur. On peut commencer avec **5** et surveiller le nombre de processus pour voir si c'est suffisant.

autovacuum_vacuum_cost_limit: La valeur par défaut est très basse. Ce paramètre permet de limiter les ressources (I/O) que le daemon autovacuum peut prendre. Augmenter ce paramètre à **3000** est une bonne idée.

Autres

lc_messages: les outils d'analyse de logs comprennent mieux les messages s'ils sont au format 'C'.

Conclusion

Avec tout cela, vous devriez être prêts à lancer votre cluster PostgreSQL en production. Il ne s'agit que d'approximations qui doivent être affinées. Rappelez-vous que l'optimisation est un travail d'orfèvre, l'amélioration de certaines requêtes peut entraîner la détérioration des performances pour d'autres requêtes.

D'autre part, il vous faudra analyser très fréquemment vos logs pour vous assurer que tout va bien. Notamment, tous les messages de type **FATAL**, **ERROR**, **WARNING** doivent être systématiquement analysés.

Enfin, n'oubliez pas de mettre en place un bon outil de backup avec une politique de rétention écrite correspondant aux DMIA (Durée maximale d'interruption admissible) et PDMA (Perte de données maximale admissible).

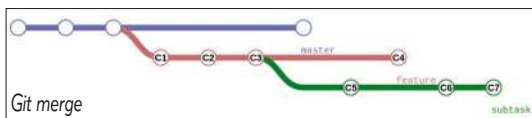
Abordez le rebase avec plus de sérénité !

Je n'en ai jamais autant appris sur Git que le soir où j'ai cassé l'historique de mon dépôt suite à un rebase chaotique et des résolutions de conflits hasardeuses. J'y ai découvert des outils et des notions que j'utilise, maintenant, au quotidien, qui me permettent de naviguer dans Git beaucoup plus sereinement, et que je vous propose d'explorer ici.

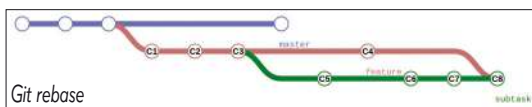
Le rebase fait partie de ces opérations Git que nous avons tendance à fuir comme la peste. La crainte de corrompre le dépôt distant de façon irrécupérable nous pousse à opter pour un merge, moins destructif, ou à laisser un de nos chers collègues s'en occuper. Si le rebase fait si peur, c'est qu'il fait écho à des notions de Git trop souvent peu maîtrisées.

Rappels sur le rebase et le merge

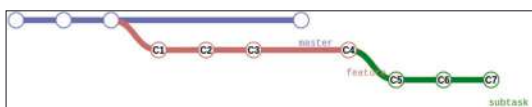
Considérons notre branche principale master, une branche feature, qui contient une feature sur laquelle travaillent plusieurs développeurs, et une branche subtask, qui contient une sous-tâche en cours de développement par l'un d'entre eux. Entre-temps, certains développeurs ont intégré leurs sous-tâches dans feature, et subtask est désormais en retard. Nous souhaitons récupérer les changements de feature dans subtask. Deux options s'offrent à nous :



En supposant que nous sommes sur subtask, `git merge feature` fusionne les deux branches en créant un nouveau commit (ici C8), appelé *merge commit*. La particularité de ce commit est de posséder 2 parents au lieu d'un (ici C4 et C7).

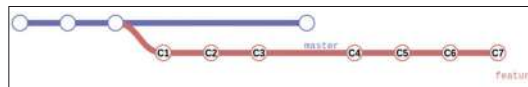


En supposant que nous sommes sur subtask, `git rebase feature` va identifier tous les commits de subtask qui divergent de feature, et les "rejouer" un par un par-dessus le dernier commit de feature. Résultat : l'historique est modifié, comme si nous n'avions jamais été en retard sur feature. Les hashes des commits C5, C6 et C7 seront d'ailleurs différents, car ils ne portent plus la même information de parent (ex: C5 a pour nouveau parent C4 et non C3). À noter qu'il faudra appeler `git push` avec l'option `-f` ou `--force` : Git refusera par défaut d'écrire la nouvelle version de subtask, car son historique a été totalement réécrit.



Le rebase apporte ici plus de clarté dans le graphe, car nous ne souhaitons pas forcément garder l'historique de branches

comme subtask. Dans cette configuration, lors du merge de subtask dans feature, Git fera par défaut un merge avec l'option de *fast-forward* (avance rapide) : comme les différences ne sont apportées que par la branche subtask, il n'est pas nécessaire d'ajouter un merge commit. Git se contentera donc "d'avancer" le pointeur HEAD de feature du commit C4 vers le commit C7. Si nous voulons absolument ajouter un merge commit, il est possible de rajouter l'option `--no-ff` afin d'empêcher le *fast-forward*.



Quand le merge/rebase est source de conflits

Il se peut parfois que feature et subtask modifient les mêmes fichiers. Git arrive la plupart du temps à fusionner les deux versions avec un *3-way merge*. En cas d'échec, nous sommes en présence d'un conflit. Charge à nous de savoir quelle version conserver : celle de feature, de subtask, des deux, d'aucune des deux.

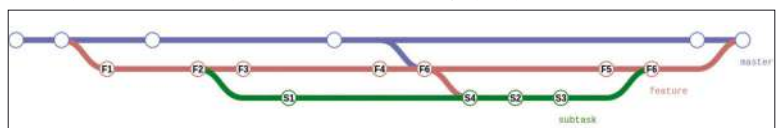
Sur ces deux concepts s'appliquent les notions de *current change* et d'*incoming change* pour caractériser les versions des branches feature et subtask, et ces termes ne désignent pas les mêmes versions suivant l'opération !

Lorsque nous sommes sur subtask et que nous exécutons `git merge feature`, nous intégrons les changements de feature dans subtask. Le *current change* correspond donc à ce que nous avons sur subtask, et le *incoming change* à ce qui provient de la branche feature.

UNE CONVENTION POUR PLUS DE CLARTÉ

Le principal avantage du merge est de pouvoir annuler l'opération d'un coup grâce à un `git revert` du merge commit. Il permet également de conserver l'historique intact. C'est pourquoi il reste utile et conseillé lors de la réintégration d'une branche enfant dans sa branche parent, surtout lorsque l'on réintègre dans master !

Ici, cela reviendrait à faire des opérations de merge pour intégrer le code de subtask dans feature, puis de feature dans master. Garder cette intégration à sens unique (et privilégier le rebase pour l'autre sens) permet de conserver un graphe plus lisible.



Un exemple de graphe avec des merge dans les deux sens : difficilement lisible !



Sonia Seddiki

Développeuse depuis 5 ans, possédant un fort penchant pour le backend et l'infra. Globe-trotteuse compulsive avec une passion secrète pour la cybersécurité et une (encore plus secrète) pour les dépôts Git bien rangés.

Lorsque nous sommes sur `subtask` et que nous exécutons `git rebase feature`, nous rejouons les commits de `subtask` par-dessus `feature` : nous démarrons à la pointe de `feature`, puis appliquons le premier commit de `subtask`, etc. Nous sommes donc dans la perspective inverse : le *current change* est-ce que nous avons à l'instant `t` sur `feature`, et le *incoming change* les changements apportés par le commit de `subtask` que nous venons d'appliquer.

Le rebase sans la peur des conflits

Il arrivera un jour où le rebase ne se passera pas comme prévu, et les conflits surviendront. Git est en général assez explicite dans ses attentes et dans les options possibles. Et bien que les choses se compliquent à mesure que l'on découvre tardivement notre erreur, il sera possible de faire machine arrière dans la plupart des cas.

Corriger des erreurs pendant le rebase

Si nous nous trompons pendant un rebase, par exemple en écrasant du code utile lors d'une résolution de conflits, alors `git rebase --abort` est notre bouton d'arrêt d'urgence ! Cette commande nous permet à tout moment d'annuler l'opération dans son entièreté afin de revenir au dernier état stable de notre branche locale.

Corriger des erreurs après le rebase local

Nous venons de rejouer votre dernier commit, le rebase s'est déroulé avec succès. Notre branche locale a désormais le nouvel historique rebasé. C'est alors que nous constatons que nous avons introduit une régression avec vos résolutions de conflit. Cette fois-ci, nous ne pouvons plus utiliser l'option `--abort` car le rebase est terminé !

Il existe une solution simple : en exécutant la commande `git reset --hard origin/<notre_branche>`, nous retrouverons l'historique de la branche distante.

Attention à bien renseigner `origin/<notre_branche>` et non `origin <notre_branche>` ! Nous ne faisons pas appel au dépôt distant, mais nous allons chercher la valeur du pointeur HEAD de notre branche remote. Celle-ci est stockée dans le dossier `.git`, plus précisément dans `.git/refs/remote/origin/<notre_branche>`

Pour fonctionner sans risquer de perdre notre travail local, il y a néanmoins quelques conditions préalables :

- Le dépôt distant doit être à jour avec **tous** nos changements sur la branche locale (autrement dit, nous avons poussé tous les commits de la branche).
- Si nous avons des changements en local qui ne sont pas dans un commit (*unstaged*), il est primordial de les stocker grâce à `git stash`, car un `hard reset` nous ferait tout perdre ! Il est également possible de faire un `soft reset` pour ne rien perdre, mais nous retrouverons aussi tous les fichiers modifiés par notre ancien rebase à l'état *unstaged*, ce qui n'est pas forcément utile.

Corriger des erreurs après avoir écrasé la branche distante

Les choses se corsent ! La branche distante est corrompue

avec nos commits, et suite au push, notre référence locale de `origin/<notre_branche>` aussi ! Il nous reste un dernier atout : les *reflogs*.

Git garde une trace des mises à jour sur la pointe des branches à l'aide d'un mécanisme appelé logs de référence ou *reflog*¹. Nous ne nous attarderons que sur la version la plus basique de la commande, `git reflog`, qui liste l'historique des derniers commits référencés par HEAD.

Rappels sur HEAD

HEAD est un pointeur qui référence un commit. Lorsque nous exécutons `git checkout feature`, HEAD va venir pointer sur le dernier commit de `feature`. Nous pouvons également spécifier un hash de commit en plein milieu d'une branche, par exemple `git checkout d0bbca`, et se retrouver en mode *detached HEAD* (lorsque HEAD ne référence plus la pointe d'une branche). `checkout`, `reset`, `merge`, `rebase`, sont autant d'opérations qui font varier le commit référencé par HEAD.

```
20fca1e85 (origin/my_branch, my_branch) HEAD@{0}: rebase -i (finish): returning to refs/heads/my_branch
20fca1e85 (origin/my_branch, my_branch) HEAD@{1}: rebase -i (pick): Seventh commit
c0ea3de97 HEAD@{2}: rebase -i (pick): Sixth commit
5c37739b7 HEAD@{3}: rebase -i (pick): Fifth commit
5dc74d0b8 HEAD@{4}: rebase -i (pick): Fourth commit
d3b499fa7 HEAD@{5}: rebase -i (pick): Third commit
0ca3efdc HEAD@{6}: rebase -i (pick): Second commit
009d2fc06 HEAD@{7}: rebase -i (pick): First commit
4f5cbc461 HEAD@{8}: rebase -i (start): checkout master
8fb819680 HEAD@{9}: checkout: moving from another_branch to my_branch
```

Un extrait de `reflog`

Chaque ligne possède une structure similaire :

```
commit référencé par HEAD (+ son nom si c'est la pointe d'une branche) | HEAD@{X}
| opération | détails de l'opération
```

HEAD@{X} signifie "position de HEAD X étapes avant sa position actuelle". La position actuelle étant ainsi HEAD@{0}, HEAD@{2} signifie nous faire retourner 2 étapes en arrière.

Reste à identifier quelle ligne correspond au début de notre rebase, et `reset HEAD` à l'étape d'avant. Ici, HEAD@{8} correspond à `rebase (start)`. Il nous faut donc exécuter `git reset HEAD@{9}`.

Nous venons de voir deux façons de rétablir une version correcte de la branche avant rebase, mais cela ne nous aide toujours pas à résoudre les conflits ! Il existe un outil qui permet de donner une meilleure visibilité lors du rebase : l'option `-i`, pour un rebase interactif.

Le rebase interactif

Imaginons un exemple des plus désastreux : quelqu'un a décidé de réécrire l'historique de notre branche commune `feature`, par exemple pour fusionner les commits C1 et C2 (c'est une pratique très risquée, à bannir absolument lors de travail sur des projets publics type open source). Après avoir `pull` le

(1) Source : <https://www.atlassian.com/fr/git/tutorials/rewriting-history/git-reflog>

dépôt distant, tenter un simple rebase sur cette nouvelle version de feature nous donnerait des conflits à coup sûr, même si subtask ne touche pas du tout aux mêmes fichiers que feature. Explication avec le rebase interactif. **Figures 1 et 2**

Depuis subtask, exécuter `git rebase -i feature` ouvre un fichier texte depuis l'éditeur de votre choix :

Code complet sur programmez.com & [github](https://github.com)

```
# Rebase dad17e8..d2980d4 onto 0e66367 (9 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
```

Un exemple de fichier de rebase interactif **Figure 3**

Ce fichier liste tous les commits de la branche subtask qui diffèrent de la branche feature, du plus ancien (première ligne), au plus récent (dernière ligne). À côté de chaque commit figure une opération de rebase. Celle par défaut est pick, ce qui signifie que le commit sera rejoué par le rebase. Nous pouvons constater que la branche subtask embarque toujours les anciens commits de feature, car ils sont foncièrement différents (tant sur leur contenu que sur leur parent) même s'ils aboutissent au même résultat. Et c'est de là que vient le conflit ! Les commits feature 1 à 3 seront rejoués par-dessus le commit feature 4, qui, chronologiquement, a déjà connu ces changements (cf graphe).

Pour supprimer ces conflits, il faut enlever ces anciens commits du fichier de rebase, soit en supprimant la ligne, soit en remplaçant pick par drop (ou d), comme illustré ci-dessous. J'ai une préférence pour préfixer par drop afin de mieux m'y retrouver lors d'un rebase plus complexe.

```
drop 50cf116 Feature commit 1
drop d0bb2f4 Feature commit 2
drop 195128c Feature commit 3
pick c4942bb Subtask commit 1
pick 5a81e9a Subtask commit 2
pick d2980d4 Subtask commit 3
```

Le rebase interactif est également très puissant pour nettoyer sa branche avant de proposer une revue de code ! Il est possible de réordonner les commits en changeant l'ordre des lignes. Attention, intervertir deux commits qui touchent aux mêmes fichiers peut soulever des conflits.

Il propose également d'autres options. Parmi les plus utiles :

- **fixup** : fusionne le commit avec son prédécesseur sans changer le message. Particulièrement utile pour les commits qui corrigent des fautes de frappe, des tests qu'on aurait oublié d'adapter, etc. Il est d'ailleurs possible, lorsque nous

exécutons `git commit`, de rajouter l'option `--fixup <hash_du_commit>`. Cela crée un nouveau commit ayant pour message `fixup! <message du commit dont on a passé le hash>`. Pratique pour facilement quel commit préfixer par `fixup` et où le positionner.

- **squash** : fusionne le commit avec son prédécesseur en modifiant le message. Si plusieurs commits d'affilée sont préfixés par `squash`, le prédécesseur du premier héritera de leur contenu, et tous leurs messages seront stockés dans la description.
- **reword** : modifier le message du commit sans en changer le contenu.

Corriger les erreurs d'un.e autre développeur.se

Si un.e autre développeur.se a introduit une régression en faisant une mauvaise résolution de conflits, mais que nous ne savons pas quand et où la régression est apparue, il est possible de nous faciliter la tâche de debug grâce à `git bisect`. Cette commande permet de remonter jusqu'au commit ayant introduit le bug en effectuant une recherche dichotomique.

1. Identifier sur notre branche un commit où votre code est fonctionnel, et un où le problème est apparu (C'est en général le cas de HEAD, sinon le problème s'est résolu tout seul !)
2. Se placer à la **racine** du projet, où se trouve le dossier `.git`
3. Exécuter les commandes suivantes
 - `git bisect start`
 - `git git bisect bad <hash_du_mauvais_commit>`
 - `git git bisect good <hash_du_bon_commit>`
4. Git nous propose alors d'étudier le commit au milieu (ici C4) afin de voir si le problème a disparu.
 - Si oui, exécuter `git bisect good`,
 - Si non, exécuter `git bisect bad`,
5. Répéter l'opération avec tous les commits que nous propose Git. Une fois toutes les options parcourues, Git nous renverra le premier mauvais commit de la branche. Nous pouvons maintenant inspecter les changements apportés par ce commit et agir en conséquence.
6. Finir l'opération avec `git bisect reset` afin de retourner sur l'ancienne position de HEAD.

Conclusion

Il existe de nombreuses solutions pour se sortir d'un rebase qui aurait mal tourné. Ces méthodes me permettent aujourd'hui de manipuler les branches en toute confiance et j'espère qu'elles vous aideront également. En cas de besoin, voici quelques ressources supplémentaires que vous pourrez toujours avoir sous la main, telles que [Dangit, Git!?!](#), ou encore [la documentation officielle](#) qui est très complète ! Si vous souhaitez approfondir votre compréhension de Git, je vous recommande également d'aller [explorer le contenu du dossier .git](#), ou encore d'analyser [les différents objets qui composent un graphe](#).

Les graphes d'exemple ont été réalisés avec la bibliothèque [GitGraphJS](#).

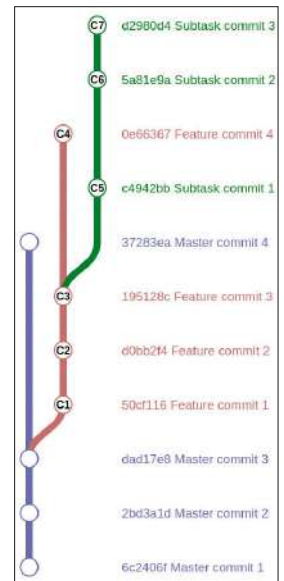


Figure 1 : Ce que nous avons avant de réécrire feature

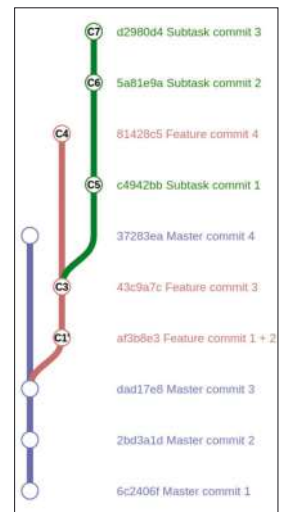


Figure 2 : Ce que nous espérons avoir après (et que nous n'avons pas du tout !)

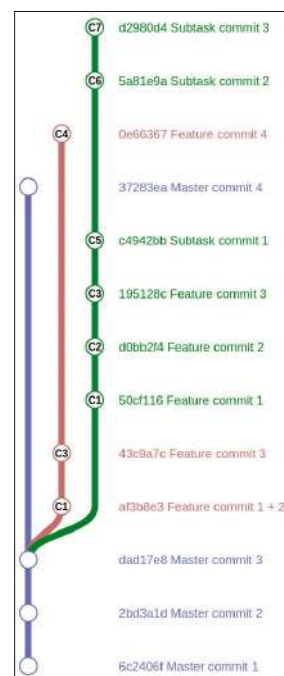


Figure 3



Marwa THLITHI

Ingénieur R&D chez
INVIVOO



Gestion des logs en Python

La journalisation des événements, appelée Logging, est une pratique très utilisée en développement, quel que soit le langage de programmation. Elle permet d'avoir un historique des événements normaux et anormaux survenus au cours du fonctionnement d'une application.

Pendant tout le cycle de vie d'une application, l'utilisation des logs est très utile et recommandée :

- Pendant la phase de développement : les logs permettent de comprendre le fonctionnement d'une application en suivant l'exécution des différentes fonctions complexes. Ils sont complémentaires au debugger
 - Pendant la phase d'intégration de l'application : ils permettent de mieux analyser des anomalies remontées
 - Pendant la phase de production : ils peuvent faciliter le diagnostic des problèmes de prod remontés par les utilisateurs.
- Au vu de l'importance des logs dans la vie d'une application, il est important d'utiliser les bonnes pratiques pour le logging et aussi des API / modules riches en fonctionnalités. Dans cet article, je vais parler du module « **Logging** » de Python et conçu pour vous donner une visibilité critique sur vos applications avec une configuration simple et minimale. Les principaux points qui seront présentés dans cet article sont :
- Les bases du module « **logging** » de Python
 - La collection et la priorisation des logs
 - La sauvegarde des logs
 - L'utilisation avancée du module Logging de Python avec plusieurs loggers

Les bases du module « logging » de Python

Le module logging est directement inclus dans les bibliothèques standard du langage. La fonction magique / clé / principale de ce module est la fonction « **basicConfig()** » qui représente un moyen très rapide pour configurer votre journaliseur – *logger*. Cependant, la manière recommandée en Python est de créer un logger pour chaque module de votre application et il peut être difficile de faire une configuration de logger par module en utilisant uniquement **basicConfig()**. Par conséquent, la plupart des applications utilisent automatiquement une configuration de journalisation basée sur des fichiers ou des dictionnaires. La fonction **basicConfig()** possède trois paramètres principaux qui sont le **niveau de sévérité** des logs – *level*, le *handler* et le *format*.

Niveau de sévérité des logs

La définition des niveaux de sévérité des logs est très importante et c'est une bonne pratique que je préconise et je recommande fortement. En effet, la génération des logs a un impact sur les performances de l'application et l'espace disque nécessaire à leur rétention. Trop de logs noient les logs importants et saturent le système, trop peu de logs nuisent à l'exploitabilité de l'application. D'où un compromis à trouver entre verbosité des logs et espace disque en définissant les niveaux de sévérité des logs en fonction de l'environnement et donc de la charge. La charge est différente d'un environnement à l'autre : sur l'environnement de développement, le développeur est seul alors qu'en production il peut y avoir plusieurs utilisateurs.

Par ordre d'alerte croissant, les niveaux de sévérité des logs disponibles dans le module logging de Python sont : **DEBUG**, **INFO**, **WARNING**, **ERROR** et **CRITICAL**. Afin d'afficher ces différents types de niveaux, le module logging fournit les fonctions suivantes : **debug()**, **info()**, **warning()**, **error()** et **critical()**. Voici un tableau récapitulatif sur les niveaux de sévérité, les fonctions correspondantes ainsi que leurs usages :

Niveau de sévérité	Fonction	Usage
DEBUG	logging.debug()	Rapport très détaillé visant à diagnostiquer un problème
INFO	logging.info()	Rapport confirmant le fonctionnement normal d'un logiciel
WARNING	logging.warning()	Rapport avertissant d'un événement / comportement inattendu
ERROR	logging.error()	Rapport sur un problème empêchant le logiciel de réaliser certaines tâches
CRITICAL	logging.critical()	Rapport sur un problème sérieux rendant le logiciel non fonctionnel

Par défaut, le niveau de sévérité est défini sur **WARNING**, ce qui signifie que le module de logging de Python filtrera tous les messages de niveau de sévérité inférieur à celui de **WARNING**, c'est-à-dire les logs de niveau **DEBUG** ou **INFO** ne seront pas affichés. Voici un premier exemple d'utilisation du module logging avec les deux niveaux de sévérité **INFO** et **WARNING** :

```
import logging

logging.warning('This warning message will be logged!')
logging.info('This is an info message')
```

Voici la sortie de cet exemple :

```
WARNING:root:Watch out!
```

C'est uniquement le log de niveau de sévérité **WARNING** qui a été affiché.

Handler

Les *handlers* vous permettent de configurer vos propres *loggers* et envoyer les logs à plusieurs endroits lorsqu'ils sont générés. Les handlers envoient les messages de logs à des destinations configurées comme le flux de sortie standard qui est la console via *StreamHandler*, un fichier, via *HTTP* ou à votre e-mail via *SMTP*. Un logger peut posséder plusieurs handlers, ce qui signifie que vous pouvez le configurer pour qu'il redirige les messages de logs dans un fichier et les envoyer également par e-mail. Comme les loggers, les handlers possèdent des niveaux de sévérité que vous pouvez définir lors de votre configuration. Ceci est utile quand vous créez

plusieurs handlers pour un même logger avec des niveaux de sévérité différents pour chaque type de sortie. Par exemple, vous pouvez afficher les logs de niveau WARNING uniquement sur la console, alors que les logs de niveau ERROR, vous les enregistrez aussi dans un fichier.

Format

Le format par défaut des messages de logs du module « logging » est `<LEVEL>:<LOGGER_NAME>:<MESSAGE>`. C'est le format de sortie de notre log de l'exemple ci-dessus. Vous pouvez personnaliser le format de vos logs pour inclure des horodatages et d'autres informations utiles pour l'identification de certains problèmes, c'est-à-dire pour le débogage.

Collection des logs avec le module « logging »

Après avoir défini dans la section précédente les principaux paramètres de configuration d'un logger qui sont le *level*, le *handler* et le *format*, nous illustrons dans cette section des exemples d'utilisation de ces paramètres avec le module **logging** et la fonction **basicConfig()**.

Exemple d'utilisation du paramètre level

Voici un exemple qui configure le paramètre *level* de la fonction **basicConfig()** :

```
import logging

logging.basicConfig(level=logging.INFO)
logging.info('This will be logged')
```

La sortie de ces lignes de code est :

```
INFO:root:This will be logged
```

Donc, maintenant tous les logs qui ont un *level* INFO et plus seront affichés.

Exemple d'utilisation du paramètre handler / filename

Comme nous l'avons indiqué dans la section précédente, nous pouvons rediriger les logs vers des fichiers avec le *handler* en utilisant les paramètres *filename* et *filemode*, sachant que ce dernier est par défaut à « append ». Ceci est illustré dans l'exemple ci-dessous :

```
import logging

logging.basicConfig(filename='log.txt')
logging.warning('This will get logged to a file')
```

Un fichier nommé « log.txt » a été créé contenant ce message :

```
WARNING:root:This will get logged to a file
```

Exemple d'utilisation du paramètre format

Nous pouvons personnaliser le format de sortie de nos logs en le configurant via le paramètre *format*. Nous pouvons passer des variables de notre programme sous forme de chaîne de caractères au format de sortie de nos logs afin qu'elles soient affichées dans nos messages de logs. Nous pouvons également utiliser certains éléments de base qui sont intégrés dans **LogRecord** et qui peuvent être facilement ajoutés au format de sortie de nos logs. Pour information, **LogRecord** est une instance créée automatiquement à partir du message de log dès que le logger enregistre un événement. Voici un

exemple de personnalisation de format en affichant des informations sur la date, l'heure, le niveau et le message à passer :

```
import logging

logging.basicConfig(format='%(asctime)s - %(levelname)s - %(message)s',
                    datefmt='%d-%b-%y %H:%M:%S')
logging.warning('User logged in')
```

Voici le résultat :

```
20-Oct-21 14:49:07 - WARNING - User logged in
```

Comme nous pouvons le constater dans notre exemple ci-dessus, `%(asctime)s` ajoute la date et l'heure de création du **LogRecord**, `%(levelname)s` affiche le niveau de sévérité du log et l'attribut *datefmt* permet de préciser le format d'affichage de la date, similaire à la fonction **datetime.strftime()** du module **datetime**.

Nous utilisons souvent les logs avec du multithreading pour suivre les exécutions de différents threads. Voici un exemple qui sauvegarde les logs d'exécution de la fonction **worker(args)** par chaque thread dans le fichier « output.log » en affichant des logs personnalisés avec le paramètre *format* :

```
import threading
import logging

logging.basicConfig(format='%(levelname)s %(threadName)s: Ceci est le %(message)s-ième message du %(threadName)s', filename='output.log', level=logging.DEBUG)

def worker(n):
    for i in range(n+1):
        logging.debug(i)

t = threading.Thread(name='thread1', target=worker, args=(10000,))
s = threading.Thread(name='thread2', target=worker, args=(5000,))
t.start()
s.start()
```

Voici un extrait du résultat sauvegardé dans le fichier « output.log » :

```
DEBUG thread1: Ceci est le 0-ième message du thread1
DEBUG thread1: Ceci est le 1-ième message du thread1
DEBUG thread2: Ceci est le 0-ième message du thread2
DEBUG thread1: Ceci est le 2-ième message du thread1
DEBUG thread2: Ceci est le 1-ième message du thread2
DEBUG thread1: Ceci est le 3-ième message du thread1
DEBUG thread2: Ceci est le 2-ième message du thread2
```

Ces logs contiennent des informations sur le *level*, le nom du thread et le message passé dans la fonction **logging.debug()**. Le nom du thread « *threadName* », comme le *levelname* et le *message*, est intégré dans **LogRecord**.

Exemple d'utilisation de variable dynamique dans les logs

Nous souhaitons souvent inclure des données dynamiques dans l'affichage de nos logs, en particulier pendant le débogage d'une application. Comme nous l'avons constaté dans les exemples ci-dessus, les fonctions de logging prennent comme argument des chaînes de caractères. Les chaînes de

caractères peuvent être facilement formatées avec des variables / données dynamiques en utilisant par exemple le style de formatage *f-string*, qui permet de garder un formatage court et simple à lire.

Voici un exemple qui montre l’affichage d’une variable dynamique qui correspond à la longueur de la variable *phrase* :

```
import logging

logging.basicConfig(level=logging.DEBUG)

phrase = 'Programmez est le seul magazine écrit par et pour les développeurs'
logging.debug(f'Votre phrase contient {len(phrase)} caractères')
```

Voici le résultat :

```
DEBUG:root:Votre phrase contient 66 caractères
```

Exemple de capture des traces de pile avec le module logging

Le module de logging vous permet également de capturer les traces complètes de la pile d’exécution d’une application. Dans ce cas, les fonctions de logging sont généralement utilisées dans un bloc de gestion d’exception, tel que le cas pour l’exemple ci-dessous :

```
import logging
a, b = 100, 0
try:
    c = a / b
except Exception as e:
    logging.error(f'Exception occurred: {e}')
```

Voici le résultat :

```
ERROR:root:Exception occurred: division by zero
```

Si nous souhaitons afficher les informations sur l’exception, nous devons mettre le paramètre *exc_info* de la fonction *logging.error()* à *True* afin de capturer ces informations. Voici le code correspondant :

```
import logging
a, b = 100, 0
try:
    c = a / b
except Exception as e:
    logging.error(f'Exception occurred: {e}', exc_info=True)
```

Voici les informations complètes sur l’exception :

```
ERROR:root:Exception occurred: division by zero
Traceback (most recent call last):
  File "exception_log.py", line 5, in <module>
    c = a / b
ZeroDivisionError: division by zero
```

Utilisation avancée du module « logging »

Nous avons bien couvert dans les parties précédentes les bases du module *logging* et en particulier la fonction *basicConfig()*, qui est un moyen rapide pour configurer un logger par défaut appelé *root*. Néanmoins, la méthode recommandée pour la journalisation consiste à créer un logger par module. En effet, au fur et à mesure que votre application évolue et devient complexe, vous aurez besoin d’un

moyen plus robuste et évolutif pour configurer un logger spécifique par module et pour capturer le nom du logger dans chaque message. Dans cette partie, nous allons découvrir comment configurer plusieurs loggers et capturer automatiquement le nom du logger.

Configuration de plusieurs loggers et capture automatique du nom du logger

Une bonne pratique de création d’un nouveau logger pour chaque module de votre application est d’utiliser la fonction intégrée *getLogger()* de la librairie « *logging* » pour définir dynamiquement le nom du logger afin qu’il corresponde au nom de votre module. Voici un exemple de création d’un logger avec cette fonction :

```
logger = logging.getLogger(__name__)
```

Dans cet exemple, nous avons mis « *__name__* » pour l’attribut nom du logger de la fonction *getLogger()* afin qu’il corresponde au nom qualifié complet du module à partir duquel cette méthode est appelée. Cela vous permet de voir exactement quel module de votre application a généré chaque message de vos logs, pour que vous puissiez les interpréter plus facilement. Par exemple, si votre application comporte un module *birthdatemodule.py*, qui est appelé depuis un autre module, *save_data_users.py*, la fonction *getLogger()* mettra le nom du logger au nom du module associé. On peut visualiser cette information quand on modifie le format du logger pour intégrer le nom « *%(name)s* » dans les messages de logs. Un exemple avec deux modules dont chacun contient un logger défini avec le même format est illustré ci-dessus. Cet exemple consiste à récupérer des informations sur des utilisateurs, qui sont le nom, prénom et date de naissance, à partir d’un fichier texte et les sauvegarder dans un fichier csv avec un format *datetime* pour le champ date de naissance. Voici le code du premier module *birthdatemodule.py* qui permet de convertir une chaîne de caractères indiquant la date au format / type *datetime*. Comme la chaîne de caractère de date précisée dans le fichier texte peut avoir plusieurs formats, dans notre code nous allons essayer de la convertir en prenant en compte deux formats possibles. Ces deux formats sont indiqués dans le premier et second bloc *try/except* de la fonction *get_birth_date()* de ce module. Hors ces deux formats, la date ne sera pas convertie et elle ne sera pas affichée dans le fichier csv qui va être généré.

```
# birthdatemodule.py
```

```
from datetime import datetime
from typing import Optional
import logging
```

```
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s %(name)s %(levelname)s: %(message)s')
logger = logging.getLogger(__name__)
```

```
def get_birth_date(date) -> Optional[datetime]:
    if date:
        try:
```

```

    return (date if isinstance(date, datetime)
            else datetime.strptime(date, '%Y-%m-%d'))
except ValueError:
    logger.debug(f'{date} is not in the format yyyy-mm-dd. Will try
to convert it to the format yyyy/mm/dd")
    try:
        return datetime.strptime(date, '%Y/%m/%d')
    except ValueError:
        logger.debug(f'Unable to convert date. Should be in format
yyyy-mm-dd or yyyy/mm/dd")
    return

```

Comme nous pouvons le constater dans notre code ci-dessus, nous gérons la conversion la date de avec des blocs de **try/except** et en utilisant des logs affichant les traitements et les avertissements importants pour le développeur / utilisateur. Voici le second module **save_data_users.py** qui permet de lire les fichiers textes précisés dans la variable **FILES** et de sauvegarder les informations extraites dans des fichiers csv avec la fonction **save_users_data()**. Comme le module **birthdatamodule.py**, ce module utilise un logger affichant le nom du module, le niveau de sévérité du message, la date et l'heure d'exécution. La fonction **save_users_data()** gère l'exception de non-existence d'un fichier avec un bloc **try/except** et en utilisant la fonction **logger.error()** pour afficher l'erreur **OSError** et la fonction **logger.warning()** pour avertir l'utilisateur de la non-existence d'un fichier.

```

# save_data_users

import re
import logging
import birthdatamodule

logging.basicConfig(level=logging.DEBUG, format='%(asctime)s %(name)s %(levelname)s: %(message)s')
logger = logging.getLogger(__name__)

FILES = ('Python_users.txt', 'C#_users.txt')

def save_users_data():
    logger.info("starting users data processing")
    for file in FILES:
        try:
            with open(file[:-4] + '.csv', 'a') as csv_file:
                print(*['Birthdate', 'User'], file=csv_file)
            try:
                with open(file, 'r') as txt_file:
                    logger.info(f'reading {file}')
                    i = 0
                    for line in txt_file:
                        data = get_users_data(line)
                        print(*data, file=csv_file)
                        i += 1
                    logger.debug(f'finish processing {i} users for the file {file}')
            except OSError as e:
                logger.error(f'error reading the file {e}')
        except:
            logger.warning(f'The file {file} doesn't exist")
    finally:

```

```

    logger.debug(f'The function is executed for the file {file}')
def get_users_data(line):
    line = line.rstrip()
    user_name = re.findall('([a-zA-Z]+)\s', line)
    d = re.findall('([0-9].*)', line)[0]
    date = birthdatamodule.get_birth_date(d)
    return [date, user_name]

if __name__ == "__main__":
    save_users_data()

```

Si nous exécutons le module **save_data_users.py**, sachant que le fichier « Python_users.txt » est accessible, mais le fichier « C#_users.txt » n'existe pas, le module **logging** générera la sortie suivante :

```

2021-10-26 11:58:35,743 __main__ INFO:starting users data processing
2021-10-26 11:58:35,744 __main__ INFO:reading Python_users.txt
2021-10-26 11:58:35,749 birthdatamodule DEBUG:1984/05/14 is not in the format
yyyy-mm-dd. Will try to convert it to the format yyyy/mm/dd
2021-10-26 11:58:35,749 birthdatamodule DEBUG:1955/10/28 is not in the format
yyyy-mm-dd. Will try to convert it to the format yyyy/mm/dd
2021-10-26 11:58:35,750 __main__ DEBUG:finish processing 5 users for the file
Python_users.txt
2021-10-26 11:58:35,750 __main__ DEBUG:The function is executed for the file
Python_users.txt
2021-10-26 11:58:35,750 __main__ ERROR:error reading the file [Errno 2] No such
file or directory: 'C#_users.txt'
2021-10-26 11:58:35,751 __main__ DEBUG:The function is executed for the file
C#_users.txt

```

Le nom du logger est inclus juste après la date et l'heure d'exécution, afin que vous puissiez voir quel module exactement a généré chaque message. Si vous ne définissez pas le logger avec la fonction **getLogger()**, les messages de tous les loggers s'afficheront avec le nom **root**, ce qui rend l'analyse des logs difficile pour faire la différence entre les messages de chaque module. Les messages qui ont été enregistrés à partir du module **save_data_users.py** auront le nom « **__main__** » affiché dans les messages du log, car ce module a été exécuté en tant que script de niveau supérieur. Les messages du logger du module **birthdatamodule.py** ont été affichés avec son nom qui est « **birthdatamodule** ».

Conclusion

Nous avons vu que la journalisation / logging est un outil très utile dans la boîte à outils d'un développeur. Cela peut vous aider à mieux comprendre le déroulement d'une application / d'un programme et à découvrir des scénarios auxquels vous n'avez peut-être pas pensé lors de la phase du développement. Dans cet article, nous avons parlé de quelques bonnes pratiques pour configurer la librairie standard « **logging** » de Python. Avec cette librairie, comme nous pouvons configurer rapidement un logger de base avec la fonction « **basicConfig()** » pour un petit projet, nous pouvons aussi configurer nos propres niveaux de sévérité de journalisation, de handlers et de formats personnalisés sur un grand projet avec la création de plusieurs loggers en utilisant la fonction « **getLogger()** ». Dans cet article, nous avons illustré des exemples d'utilisation basique de logger et aussi des exemples d'utilisation avancée avec la création de plusieurs loggers ayant des formats personnalisés.



CommitStrip.com

Directives de compilation

PROGRAMMEZ!

Programmez! n°250 - Janvier - Février 2022

Seigneur Sith niveau directeur des rédactions :
François Tonic - ftonic@programmez.com

Rédactrice en chef du numéro 250 : Dorra Bartaguiz

Contacter la rédaction : redaction@programmez.com

Ont collaboré à ce numéro : L. Adam (Zdnet), CommitStrip

Les contributeurs techniques

F. Letrange,	C. Devoyod,	C. Hermande
R. Alula,	M. Tazua,	Pajuelo,
A-L Gaillard,	M. Avomo,	P. Seck Ndedi,
H. Goma,	R. Alula,	G. Bianchi,
P. Garric,	Armelle Youmbi,	L. Tachet,
S. Dufour,	C. Bibi,	A. de Amaral,
M. Viktoriia,	J. Mehdad,	L. Avrot,
S. Aziki,	L. James,	M. Thlithi
E. Carli,	C. Doolaege,	
C. Duvigneau,	S. Seddiki,	

Couverture
© DISNEY - TRON

Maquette
Pierre Sandré

Marketing - promotion des ventes
Agence BOCONSEIL - Analyse Media Etude
Directeur : Otto BORSCHA
oborscha@boconseilame.fr
Responsable titre : Terry MATTARD
Téléphone : 09 67 32 09 34

Publicité
Nefer-IT
Tél. : 09 86 73 61 08
ftonic@programmez.com

Impression
SIB Imprimerie, France

Dépôt légal
A parution

Commission paritaire
1225K78366

ISSN
1627-0908

Abonnement

Abonnement (tarifs France) : 49 € pour 1 an,
79 € pour 2 ans. Etudiants : 39 €. Europe et Suisse :
55,82 € - Algérie, Maroc, Tunisie : 59,89 € - Canada
: 68,36 € - Tom : 83,65 € - Dom : 66,82 €.

Autres pays : consultez les tarifs
sur www.programmez.com.

Pour toute question sur l'abonnement :
abonnements@programmez.com

Abonnement PDF
monde entier : 39 € pour 1 an.
Accès aux archives : 19 €.

Nefer-IT
57 rue de Gisors, 95300 Pontoise France
redaction@programmez.com
Tél. : 09 86 73 61 08

Toute reproduction intégrale ou partielle est interdite
sans accord des auteurs et du directeur de la
publication. © Nefer-IT / Programmez!,
janvier 2022.