

Projet Professionnel Dévops

Sayfoutdinov Said-Timour

2023/2024

*Drivemanager*

Outil de gestion des taxis médicalisés

Rapport  
mensuel  
de lʼentreprise

**P**

our commencer, touchez ou cliquez sur ce paramètre fictif, et débutez la saisie. Vous pouvez voir et modifier ce document sur votre Mac, iPad ou iPhone, ou sur iCloud.com.

Vous pouvez modifier du texte et des polices et ajouter de beaux graphiques. Utilisez des styles de paragraphes pour un document à lʼaspect uni. Par exemple, le style Corps est utilisé ici. Vous pouvez le modifier dans lʼonglet Texte des commandes Format.

Remerciement

*Ce dossier représente l'aboutissement de mon année d'études chez Simplon (à distance). Je tiens à remercier l'ensemble des personnes m'ayant accompagné dans cette tâche.*

* *Patrice Guillaume (DSI), Responsable d'activité Service Informatique (AURA), Tiphaine Hanry (Simplon) et Leslie Bitene-Verrier (Simplon), Responsables de la promo, pour leur accompagnement et leurs conseils.*
* *DSI pour m'avoir accepté et accompagné malgré les difficulté possiblement contraignant et un profil peu orthodoxe.*
* *Simplon Grenoble pour m'avoir accepté malgré les aménagements peu compatible avec la formation.*
* *Bien sûr je ne peux omettre ma reconnaissance envers les enseignants Thomas Laforge et Anthony Youssef pour leur implication dans notre réussite, leur organisation et leur enseignement de façon général*
* *Et pour finir à mes proches pour leur soutien moral et leurs conseils tout au long de cette formation.*

*A toutes ces personnes, je présente mes remerciements, mon respect et ma gratitude.*

Sommaire

Introduction. 4

Mise en place de la méthodologie devops. 5

Optimisation du Processus de Développement. 6

*Gestion du Versionnement, Collaboration, Assurance Qualité et Intégration Continue 6*

Bilan du projet. 15

*Problématiques rencontré. Amélioration envisagés. 15*

*Discussion. 15*

Conclusion 17

Bibliographie 17

Introduction.

Présent écrit, est le fruit du travail, fait pour la certification DevOps, dont nous préparons dans la cadre de la formation préparant au titre de niveau II, s’intitulant, Concepteur Développeur d’Application, dans un environnement DevOps.

A la fin de cette introduction, nous allons présenter notre projet « chef d’œuvre » de façon sommaire, afin donner une vison plus global sur le travail que nous menons.

Ensuite, dans la première partie du dossier, nous allons exposer la mise en place de la méthodologie DevOps. Egalement nous aborderons qu’est ce que peut apporter cette approche.

Dans la suite de l’écrit, nous allons exposerons le pipeline que nous suivons et comment nous avons mis en place une infrastructure programmable. De ce fait, nous présenterons les outils de travail.

Notre projet porte sur la création d’un outil pour les entreprise des taxis médicalisé/VSL. Pour être plus précis, ces sont des entreprises agrémenté et sont spécialisées dans le transport des malade, dans le cadre des traitements ou des consolations médicale. L’outils que nous souhaitons proposer, est un outils de gestion centralisé des courses, clients et chauffeurs.

Au moment de la production de cet écrit, le travail sur le projet avance de plus en plus vite car le passage de titre se rapproche. A ce moment même, nous commençons le travail sur le front-end et l’application pour les chauffeurs. Cependant les discussions sur la façon de travail ne sont pas terminé. Toutefois nous faisons notre mieux pour accomplir ce dossier., en donnant les exemples issu de notre projet et nous essayons d’avance au mieux pour cela également.

Mise en place de la méthodologie devops.

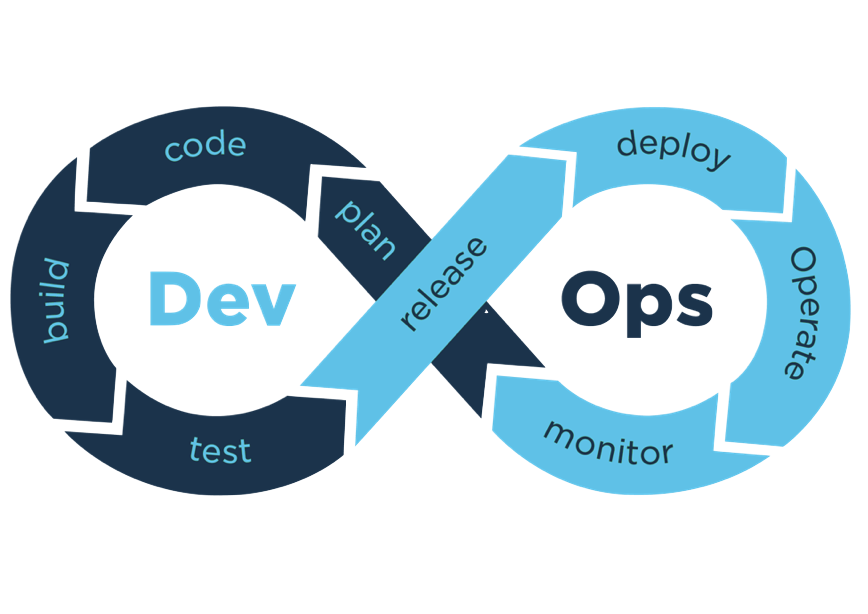


Fig. 1

Dans cette partie, nous allons aborder les outils que nous avons utilisé pour la mise en place de l’approche DevOps et nous argumenterons sur nos choix.

Les intitulés des sections, qui vont suivront, sont des reformulations des intitulés des compétences listé dans le référentiel « Certification DevOps ».

Il serait probablement possible de constater, que certains arguments, sont des rappels directes des compétences que nous voulons valider par le biais de l’outil ou approche utilisé.

Globalement notre façon de travailler est basé sur l’approche, dite, déclarative, car nous avons définit les environnements de travail avant de commencer notre travail.

Egalement nous sommes parti du postula suivant:

« *C’est le cumul d’itérations et de feedback sur le code qui vont nous permettre d’améliorer sa fiabilité, sa stabilité, sa performance et sa lisibilité. »*

Optimisation du Processus de Développement.

Gestion du Versionnement, Collaboration, Assurance Qualité et Intégration Continue

Pour commencer l’exposition du travail effectué, nous avons décidé de commencer par la présentation de la gestion du versionnement du code. Car, l’outil que nous avons adopté nous était déjà familier.

Ainsi, nous poser trop questions nous avons décidé d’utiliser Git et GitHub, dans un premier temps, ce choix vient du fait que, tout au long du parcours d’apprentissage nous avons utilisé ces outils.

Dans un second temps, ayant prévu utiliser GitHub Actions pour l’automatisation, le GitHub est, juste, incontournable. Alors nous nous sommes demandé:

* *Pourquoi changer l’équipe, qui gagne?*

Également, dans le future, au moment de l’intégration des nouveaux collaborateur, nous sommes certains qu’ils sauront l’utiliser.

En utilisant GitHub, pour ce projet, nous avons fait le choix de créer plusieurs repositorys. C’est à dire, nous allons nous servir d’un repository pour le back-end, d’un autre pour le front-end et d’un autre pour le code de l’application mobile des chauffeurs.

Nous pensons que cette approche, nous convient le plus, car l’application de standardiste et l’application de chauffeurs ne sont pas similaire et nous risquons pas nous retrouver face à la question:

* Combien de fois tel ou tel composant est-il utilisé?

Aussi dans le future quand ce projet prendre l’ampleur, il sera plus facile d’attribuer les permission d’accès aux collaborateur.

Afin de mieux articuler la présentation, le fonctionne de l’intégration continue sera aborder dans la suite de cette partie. Avant cela, dans les paragraphes qui précéderons la présentation de notre pipeline, nous exposeront l’approche de gestion de versionnement du code, que nous appliquerons à nos trois référentiels. Dans la figure 2, nous avons voulu illustrer le modèle de gestion de branche que nous avons adopté.

Pour les trois repositorys, il était décidé d’utiliser trois branches principales, « Dev », « Main » et « Production ». Ces branches sont permanentes, alors que la branche « Feauture » va être utilisé pour la mise en partage de nouvelles fonctionnalités ou de correctifs. Ce ne sera pas une branché permanente. Elle sera supprimé, après chaque fusion avec la branche « Dev » et recréée au besoin.

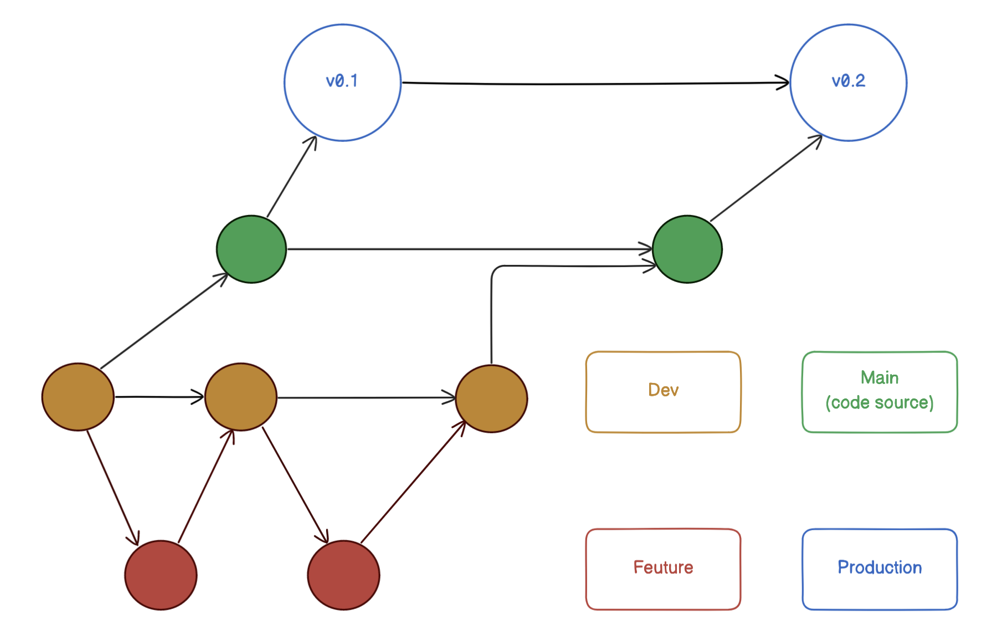


Fig. 2

La branche « Dev », sera utilisée pour faire passer les premiers tests automatisé et puis être confronté à un « buil », dans un environnement autre que celui de la machine du développeur, avant d’être fusionné avec la branche « Main ».

La branche « Main » aura plus au moins le même fonctionnement que « Dev ». C’est à partir de cette branche que nous allons envoyer le code dans la branche « Production ». La branche « Production », sera utilisé pour le déploiement de notre repository.

Afin d’avancer l’exposé du travail effectué dans le cadre du projet et de la certification DevOps, nous présenterons la pipeline adopté. De ce fait, cette section du chapitre présentera nos compétences du contrôle du code produit, de l’automatisation des tests dans un environnement configuré pour la CI et CD. Et après cela, nous aborderons la veille technologique permettant le suivi de l’application en temps réel. Ainsi pour mieux expliquer cette partie de travail, nous avons décidés de joindre l’illustration de pipeline (fig. 3). Cependant étant donné que nous avons décidé de séparer en trois repository différents notre projet, il nous fallait « trois » pipeline pour chaque repo. Nous avons décidé de joindre ici la pipeline de Front-end, la seul différence que nous aurons avec celle pour le back, c’est qu’il n’y aura pas de tests end-to-end.

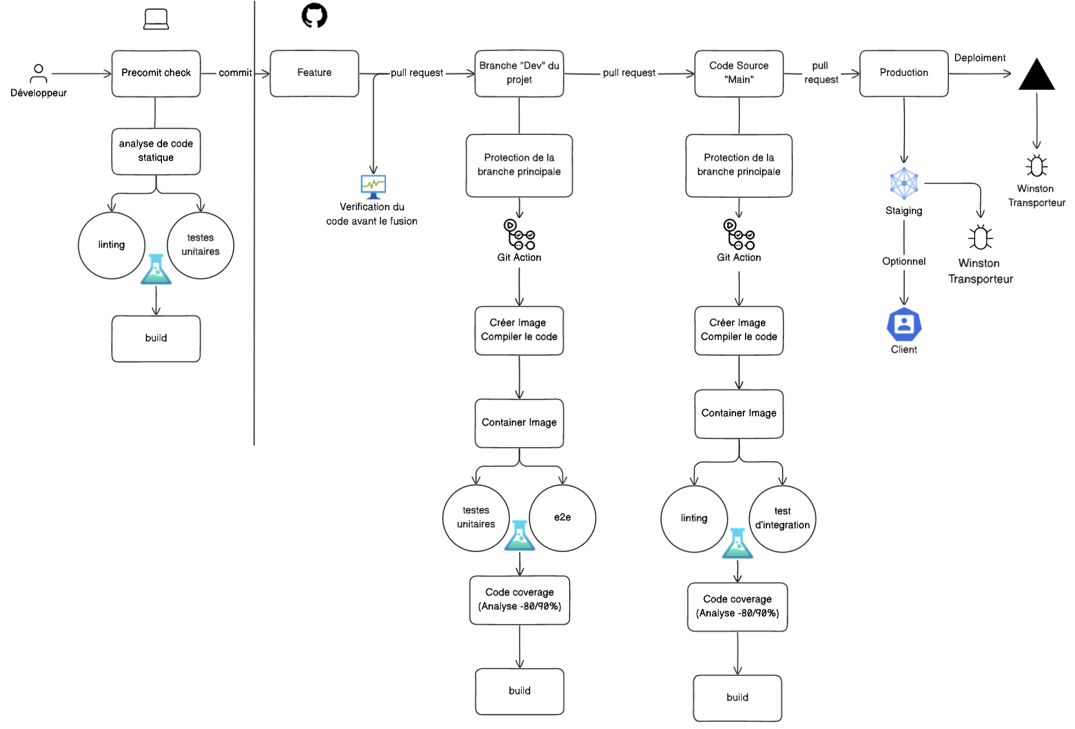


Fig. 3

Sur cette illustration, il est possible de constater que, nous avons fait une séparation, en deux, entre la machine du développeur et GitHub, avec le trait vertical. Également, il est possible de voir que nous avons plusieurs série de tests pour chaque branché permanente.

Cette mise en place de tests, était inspiré par l’article s’intitulant « *Traversée de la pyramide d’automatisation des tests »*[[1]](#footnote-1) et l’illustration de la pyramide qui est présente dans cet article et que nous avons intégré dans ce dossier (fig. 4). Cette illustration se lit du bas vers le haut.

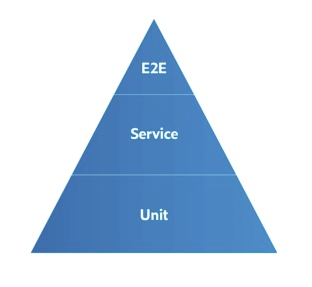


Fig. 4

Sur cette illustration, nous pouvons constater que les auteurs présentent les tests unitaire comment des testes de « base » et donc il doivent être fait en premier, une fois ces tests passé nous passons à la couche « service » et elle fait référence aux teste d’intégration. A la fin nous retrouvons la couche e2e, qui fait référence aux tests end-to-end, donc ce sont les tests qui vont vérifier le fonction de l’application dans sa globalité.

Passons maintenant à la lecture de notre pipeline. Dans un premier temps, nous avons fait le choix de mettre en place des precommit checks, autrement écrit, avant de envoyer le travail vers le GitHub, nous allons faire les tests à l’issue desquels nous allons pouvoir envoyer notre travail dans la branche « Feuture », uniquement si les tests et le build sont passés sans erreurs. Dans le cadre du precommit check nous souhaitons que les développeurs dans le future et nous même fassions des tests de linting et des tests unitaires. Dans les figures 6(package.json), 7 (eslintrc.json) et 8 (.prettierrc), nous exposons nos fichiers de configuration, pour faire fonctionner les tests de linting.

Les tests de linting vont être assuré par ESLint et Prittier. Les raisons pour lesquelles, nous les avons choisi, sont assez simple. Nous utilisant cette combinaison d’outils car ESLint va vérifier la syntaxe et des variables fantômes « non utilisées » et Prettier va s’occuper uniquement du format stylistique car nous voulons avoir un code formaté horizontalement et verticalement comme sur la figure 5. Nous considérons ce point, comme important car, il permet une meilleur lisibilité au code, puis que notre but est de pourvoir travailler facilement avec le code. Pour cela nous nous imposant, également, la convention de nommage des variable, « camelCase », mais aussi nous allons nous imposer de nommer des variables de façon explicite, afin de comprendre leur utilité dès la lecture. Puisque un code autodocumenté est un code qui peut lu compris, maintenu rapidement par les développeurs extérieur



Fig. 5

Pour argumenter nos choix de ces outils. ESLint, étant un outils configurable, il nous permettra avoir une cohérence syntaxique assez facilement. Egalement, c’est un outil qui nous donnera possibilité d’identifier des bugs, directement, à l’écriture, plutôt qu’en « buildant » l’application. Le choix d’utiliser Prettier était assez directe car dans les articles que nous avons pu lire, la combinaison des ces deux outils est promu et encouragé. Aussi la facilité de configuration en soi et en complément de ESLint nous a séduit.



Fig. 6

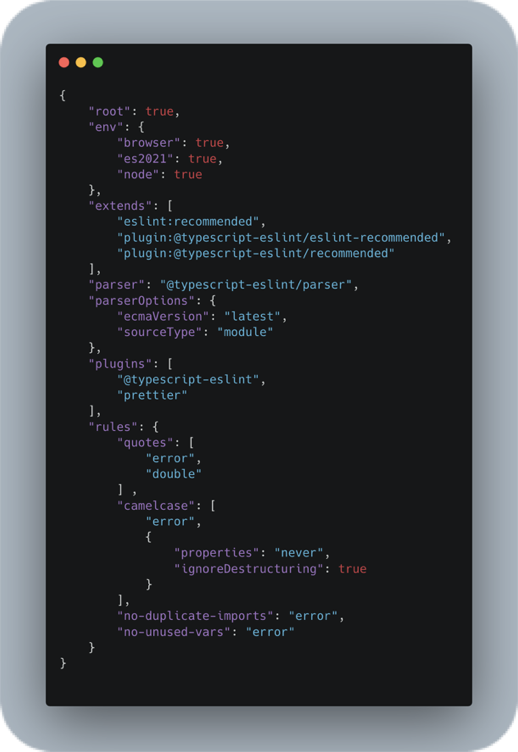


Fig. 7

Pour terminer cette partie, portant sur linting, nous allons aborder rapidement l’outil que nous voulons, en tant que générateur de documentation. Cette tâche peu amusante, sera fait par Storybook, car c’est un outil « open source » qui permet de créer des composants isolés à des fins de documentations, et il est compatible avec les frameworks, front-end, utilisés.

Dans la suite de precommit check, nous lançons les testes unitaires localement pour nous assurer du bon fonctionnement de chaque composant du code. Pour cette étape, nous utiliserons Jest en le combinant avec supertest. Jest est un framework de teste JavaScript maintenu par une communauté de contributeurs open source et d'employés de Facebook. Nous l’avons choisi car il est compatible avec TypeScript et il est capable de compiler le code TS avant d’exécuter les tests, automatiquement. Également, cet outil a une documentation abondante et dû à son large adaptation il bénéficie de nombreux tutoriels, facilitant ainsi sa maîtrise.



Fig. 8

Apres la validation des tests unitaires, nous procéderons à un build en local pour vérifier la compilation du code. A ce stade, nous utilisons le TypeScript Compiler (tsc), puis que c’est un outil officiel, il conçu pour être conforme aux normes et spécifications de TypeScript (*alors nous nous sommes demandé: pourquoi changer l’équipe qui gagne?*). Une fois l’étape de precommit check est accompli, nous poussant notre branche local « Feature » vers la branche distante, pour que l’utilité du code soit vérifié, après la validation le code va être ajouté au code se trouvant dans la branche « Dev ». C’est à ce moment la que GitHub Action va prendre le relais, grâce à la configuration du fichier YAML dans le gitflow (fig. 9). De façon général nos fichier YAML se rassemble donc c’est pour cela que nous avons joint un seul fichier.



Fig. 9

GitHub Actions procédera à compilation du code pour faire des tests unitaire et des tests end-to-end (si c’est le repo du front de l’application). Encore une fois, nous aurons recours aux tests unitaire avec Jest, seule différence, c’est que, cette fois le lancement est automatisé. Pour les tests end-to-end, nous allons utiliser Playwright. Ce choix était fait suite à lecture du témoignage de Diaz, L. (2023). *Why I switched from Cypress to Playwright. Medium.com.* A la fin des tests GitHub Action va refaire un build pour nous assurer un code fonctionnel.

A ce stade notre branche « Dev » ne doit pas avoir d’erreur, pour que nous puissions continuer la CI et évoluer vers la CD. Donc, il nous restera encore une batterie de tests. Ces tests sont les suivant, linting et les tests d’intégration. Pour linting, nous utiliserons les même outils qu’en local. Pour les tests d’integrations nous avons choisi d’utiliser SuperTest avec Chai. Une fois les tests seront validé, cette branche enverra le code vers la banche de production. À son tour la branche « Production » aura deux fonction assez importante, déployer et envoyer vers DockerHub, son code, pour qu’il soit testable dans un environnement de staiging et éventuellement présenté au client pour un retour.

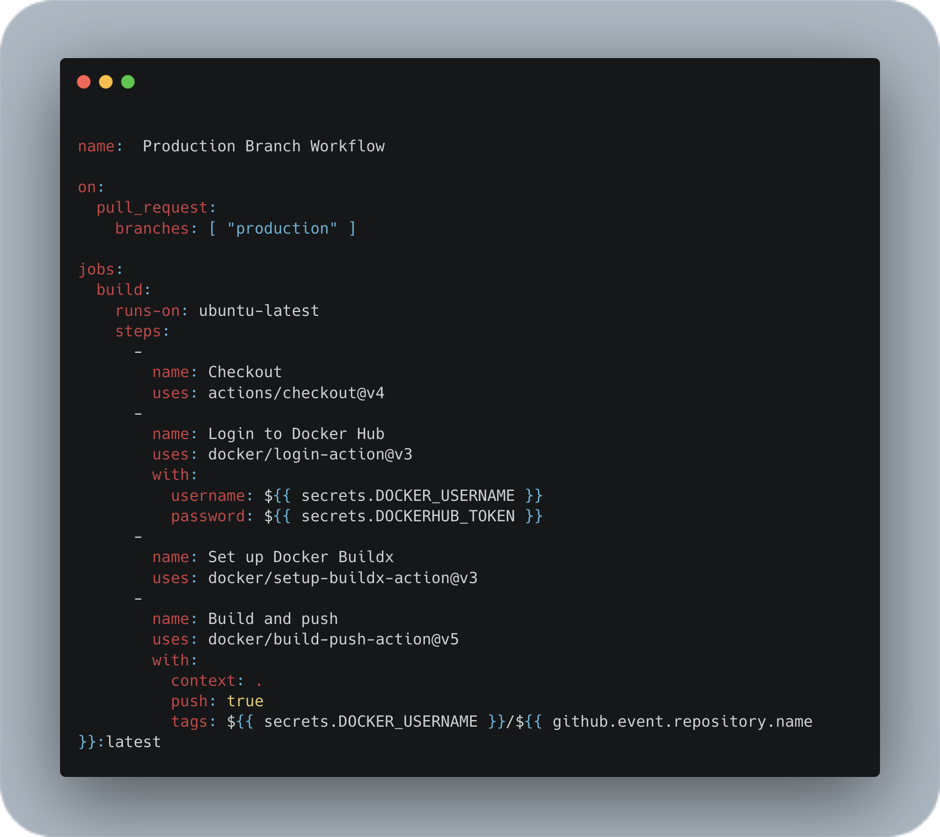


Fig. 10

Pour suivre notre application de manière quotidienne nous allons mettre en place « Winston », un outil de journalisation. Il va nous aider à tracer des éventuels erreurs.

Bilan du projet.

Problématiques rencontré. Amélioration envisagés.

Comme il était dit dans l’introduction, le travail sur le dossier était tombé juste au moment nous travaillons sur le projet, ce qui fait nous ne sommes complètement sûr de nos choix, comme par exemple ce dossier et l’élaboration de pipeline nous fait douter sur le choix d’utiliser plusieurs repositorys, car nous n’avons pas encore assez de connaissances pour mener à bien les tests dans ces conditions. Et donc nous réfléchissons encore sur ce point. Par conséquent nous pensons de un repos unique pour tout le projet et refaire les tests de pipeline. Cela nous permettra comparer les deux approches et choisir la meilleur, certainement si nous changeant l’approche il nous faudra revoir notre configuration de pipeline.

Autre soucis, que nous avons pu rencontre, sont lié au choix des framework pour les tests, car ne connaissant pas tous les outils existant, il est facile de se tromper d’outil. Pour surmonter ce problème, nous nous somme résolu à lire et regarder plus de témoignage et article possible, ce qui a bien enrichie notre bibliographie. Bien sur il peu probable que nous changeant ses outil dans le cas du changement de gestion de notre code sur GitHub, toutefois nous continuons la recherche au sujet des tests unitaire, d’intégration et d’end-to-end.

Dans le cadre de certification nous devions mettre en place des micro-services, cependant nous ne l’avons pas fait car nous n’avons pas trouvé de réel utilité, étant donné que le projet est nouveau, il n’est pas était confronté à la « vie réelle » et donc nous ne pouvons pas savoir avec exactitude quels sont les services qui vont être surchargé et à quel endroit il vaut mieux scier l’application. Également nous avons lu un

Discussion.

Au cours de nos recherche pour ce projet, nous avons lu plusieurs témoignage et articles sur la méthodologie DevOps et les outils. Le fait de voir différentes proposition, nous constatent remis en question, ce qui fait que même au cours de l’écriture du dossier nous ne sommes pas sûr des choix, mais nous avons décidé quand même de les tester et voir si cela fonctionne ou pas. Cette approche nous oblige de travailler beaucoup et vite pour être dans les temps. Il y a aussi des bons côté, les témoignages que nous avons exploré, nous ont permis de rejeter certain propositions et garder d’autre. Comme par exemple, nous n’avons pas adopté l’utilisation de « Husky Precommit Hook » alors que nous trouvons que cet outils est assez pratique. Nous ne l’utilisons pas car dans notre infrastructure automatisé, il serait préférable que nous regardions quand même le code, car cela nous permettre à mieux intégrer les conventions, il est possible que cet outil ferait partie de Stack Technique dans le future proche, une fois que nous aurons plus d’aisance avec notre projet.

Conclusion

Bibliographie

A successful Git branching model. (s. d.). nvie.com. https://nvie.com/posts/a-successful-git-branching-model/

Atlassian. (s. d.). Principes des microservices | Atlassian. https://www.atlassian.com/fr/microservices

Benefits of a DevOps environment | MuleSoft. (s. d.). MuleSoft. https://www.mulesoft.com/resources/api/devops-environment-benefits

Bogdan Stashchuk. (2021a, novembre 4). Back-end development and APIs - FreeCodeCamp tutorial [Vidéo]. YouTube. https://www.youtube.com/watch?v=hHLmb3OD7Mo

Bogdan Stashchuk. (2021b, novembre 4). Back-end development and APIs - FreeCodeCamp tutorial [Vidéo]. YouTube. https://www.youtube.com/watch?v=hHLmb3OD7Mo

Catoire, L. (2023, 16 mai). Quelle architecture de projet choisir entre micro-services et monolithe modulaire ? Efficience IT. https://www.itefficience.com/article/quelle-architecture-de-projet-choisir-entre-micro-service-ou-monolithe-modulaire

DevOps Journey. (2023, 17 octobre). How to design a modern CI/CD pipeline [Vidéo]. YouTube. https://www.youtube.com/watch?v=KnSBNd3b0qI

Losoviz, L. (2023, 12 septembre). Mono-repo vs multi-repo : avantages et inconvénients des stratégies de dépôt de code. Kinsta®. https://kinsta.com/fr/blog/mono-repo-vs-multi-repo/

McKendrick, J. (2017, 20 avril). Pourquoi les microservices ne sont peut-être pas faits pour tout le monde. ZDNetFR. https://www.zdnet.fr/amp/actualites/pourquoi-les-microservices-ne-sont-peut-etre-pas-faits-pour-tout-le-monde-39851458.htm

Monolithic vs. Microservices : Why decoupled and headless architectures are the future | Contentstack. (s. d.). https://www.contentstack.com/cms-guides/monolithic-vs-microservices-cms-architectures

Wallen, J. (2022, 7 novembre). How to build a docker image and upload it to Docker Hub. TechRepublic. https://www.techrepublic.com/article/how-to-build-a-docker-image-and-upload-it-to-docker-hub/

What Is an API Gateway ? (s. d.). Nginx. https://www.nginx.com/learn/api-gateway/

1. Cet article est peut être lu à l’adresse suivante: https://fr.parasoft.com/blog/testing-automation-pyramids-for-software-development/ [↑](#footnote-ref-1)