# ETUDE DE CAS

# SUJET DEVOPS: PROJET D'OBJET CONNECTÉ (IOT)

SAYFOUTDINOV SAID-TIMOUR

# **Remerciement**.

*Ce dossier représente l'aboutissement de mon année d'études aux Simplon en cours 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

Présentation de l'entreprise.....4

Problématique. .....4

Objectif de modernisation.....5

**Compréhension du besoin. Les enjeux de la mise en œuvre de la méthodologie DevOps.....6**

KPI, SLI et SLA dans la Méthodologie de Projet.....6

La Modernisation du Workflow. .....7

Philosophie DevOps et Mono-repos. .....7

Documentation et Code Autodocumenté. .....9

Pipeline du Projet. .....11

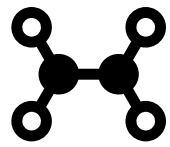
**Bilan de l'étude. .....16**

Récapitulatif. .....16

Discussion. .....16

Conclusion. .....18

Bibliographie. .....19



# Introduction.

Dans un paysage technologique et commercial en mutation constante, les organisations doivent revoir leur approche du développement et du déploiement de logiciels. L'adoption de l'approche DevOps offre une solution intégrée pour garantir l'efficacité, la qualité et la stabilité des projets, tout en favorisant une culture de collaboration.

Ce dossier examine l'application des principes DevOps dans une entreprise spécialisée dans l'IoT. Nous analyserons les défis rencontrés par cette entreprise et explorerons des solutions pour atteindre ses objectifs de modernisation. Nous mettrons en lumière les besoins spécifiques de ce projet, soulignant l'importance de l'approche DevOps pour optimiser les processus de développement et de déploiement.

À travers cette étude, nous proposerons des recommandations et des stratégies pour une mise en œuvre de l'approche DevOps, adaptée aux objectifs et besoins.

# Présentation de l'entreprise.

L'entreprise se spécialise dans le développement d'instruments de diagnostic dédiés au contrôle de la qualité. Elle se distingue par la fourniture de produits de haute qualité et jouit d'une solide réputation dans les domaines de la mesure de l'air, de l'eau, du sol et de la recherche sur la biodiversité.

Elle a conçu un logiciel de bureau centralisé pour la gestion unifiée de ses produits, simplifiant ainsi la configuration, la mise à jour et l'étalonnage des instruments. Récemment, l'entreprise a initié le développement d'une application mobile permettant aux clients de configurer leurs appareils via leurs smartphones Android et iOS.

Le langage de développement privilégié par l'entreprise est principalement TypeScript et JavaScript, tandis que React Native et Electron assurent respectivement la création d'applications mobiles et de bureau.

En permanence à la recherche de nouveaux talents, elle recherche particulièrement des développeurs JavaScript full-stack seniors, dotés d'une expertise approfondie en React Native et Electron. Structurée en cinq équipes de développement, chacune dirigée par un product owner, un scrum master, un lead développeur et un QA.

L'équipe DX (Developer Experience) est chargée de mettre en place des outils de développement, de dispenser des formations et de promouvoir les meilleures pratiques.

# Problématique.

De nombreux petits projets sont stockés dans des repository distincts. Certaines fonctionnalités requièrent des modifications dans plusieurs repository. L'objectif est de gérer ces modifications de manière centralisée et de les déployer de façon coordonnée.

Actuellement, lorsqu'un repository est modifié, une pull request est soumise, suivie de l'attente de la validation, puis d'une release, et encore d'une validation. Ensuite, le déploiement est effectué, suivi d'une nouvelle attente de validation. Ce processus doit ensuite être répété pour les autres repository impactés par la modification. Par exemple, si le firmware du capteur d'air est modifié, les ajustements doivent également être apportés à l'application mobile, à l'application de bureau, au back-end et au front-end web.

Des difficultés émergent également dans l'intégration des développeurs sur la partie front-end. Certains réclament des documentations ou des tests, mais le QA ne réalise que des tests physiques et n'a pas le temps d'effectuer des tests automatisés. Sur le back-end et l'électronique, des tests unitaires sont utilisés, mais rien n'est fait pour le front-end.

Enfin, l'analyse des erreurs rencontrées par les utilisateurs pose problème. Malgré l'efficacité de nos outils de monitoring, nous souhaitons obtenir des informations plus précises sur le code défectueux, les retours d'erreurs, etc. Des cas liés aux versions de navigateurs, systèmes d'exploitation et firmwares ainsi que des problèmes de connexions nécessitent des informations plus détaillées.

## Objectif de modernisation.

- Améliorer le  workflow de développement (éviter les npm link et les PR de sous projets).
- Améliorer la qualité du code et d'un code beaucoup plus homogène.
- Eviter les régressions.
- Automatiser le plus de choses possibles.
- Avoir une documentation à jour.
- Rassembler tout les projets dans un seul repository.

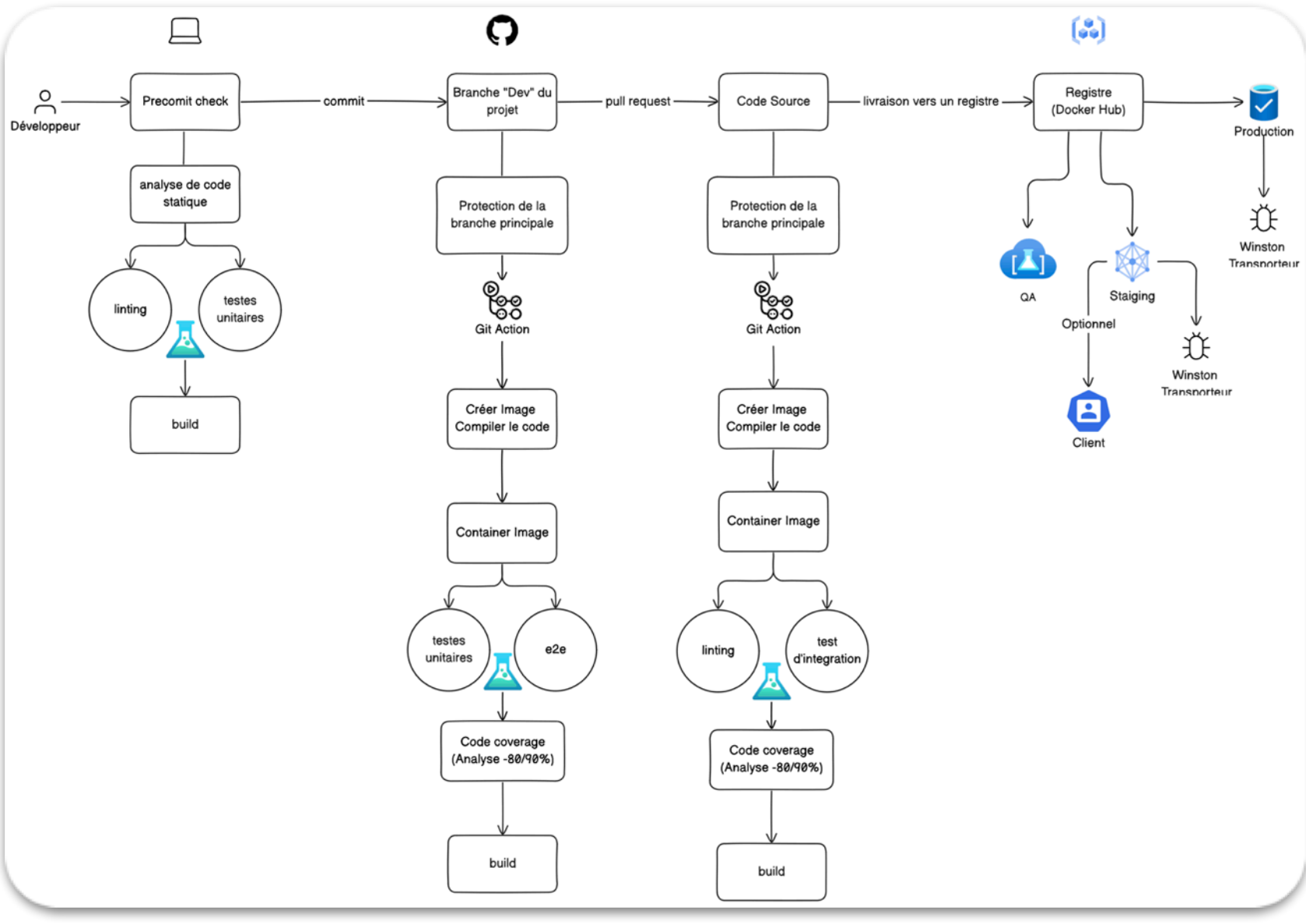
# Compréhension du besoin. Les enjeux de la mise en œuvre de la méthodologie DevOps.

## Solutions apportés. Justification des choix.

Pour répondre à la problématique soulevée, voici notre proposition de solution.

## KPI, SLI et SLA dans la Méthodologie de Projet.

La mise en place d'une méthodologie efficace constitue un pilier fondamental du succès des projets, tant dans leur gestion que dans leur exécution. Dans ce contexte, la définition précise, selon notre avis, des indicateurs comme (KPI[1]), des objectifs de niveau de service (SLI[2]) et des accords de niveau de service (SLA[3]) revêt une




Pipeline. (Fig. 1)

*Le diagramme peut être vu à l'adresse suivante: https://app.eraser.io/workspace/eUuXTC7HOgWm6g2IlzZq?origin=share*

[1] KPI en anglais ou indicateur de performance en français est une mesure quantitative qui vous permet de suivre la progression de votre équipe ou organisation au regard de vos objectifs commerciaux clés.

[2] Les SLI sont les indicateurs factuels de la disponibilité du service. Ils représentent l'état du service à un moment donné. Ce sont ces indicateurs que l'on mesurera avec le système de monitoring pour prouver le taux de disponibilité d'un service donné.

[3] SLA est un engagement entre deux parties. Cela peut être entre deux entreprises. Le SLA va définir l'engagement sur la disponibilité du service.

importance cruciale. Ces éléments offrent une feuille de route permettant de mesurer, d'évaluer et de garantir la qualité des produits ou services développés.

La société, étant reconnue pour son expertise, et dans un contexte DevOps, où l'accent est mis sur l'automatisation, la collaboration et l'amélioration continue entre les équipes de développement et d'exploitation; la définition et la mesure des KPI, SLA et SLI sont important pour garantir la qualité.

Pour le calcul des KPI pour le SLI et le SLA, il est important de choisir des mesures qui reflètent la performance et la disponibilité des services offerts par les applications développées. Ainsi dans le cadre du projet, nous proposant, récupérer, différentes mesures pour le calcul de KPI à partir de SLA et SLI:

SLI :

1. Taux de Disponibilité :
   Mesuré à partir du temps où les services sont disponibles pour les utilisateurs finaux.
2. Temps de Réponse Moyen :
   Calculé à partir du temps écoulé entre la réception de la requête et l'émission de la réponse.

SLA :

1. Objectif de Disponibilité :
   Définit le niveau de disponibilité attendu pour les services, généralement exprimé en pourcentage.
2. Temps de Résolution des Incidents :
   Spécifie le délai maximal acceptable pour résoudre les incidents ou les problèmes signalés.

La formule pour calculer le KPI pourrait être la suivante:

$$\text{KPI} = \frac{\text{SLI}}{\text{SLA}} \times 100$$

Cette formule exprime le rapport entre le niveau de service réel (SLI) et le niveau de service attendu (SLA), généralement exprimé en pourcentage pour obtenir une mesure comparative plus intuitive.

# La Modernisation du Workflow.

## Philosophie DevOps et Mono-repos.

Pour amorcer le processus de modernisation, notre proposition préliminaire vise à regrouper tous les projets au sein d'un seul repository, afin de centraliser la gestion des modifications. Cette démarche requiert la mise en place d'une pipeline (CI/CD) rigoureuse, pour garantir une efficacité optimale. Cela suppose que tous les développeurs suivent des normes communes.

Ainsi cette mise en place de la centralisation des modifications, d'une pipeline CI/CD rigoureuse et l'adoption de normes communes, favorisera une approche DevOps en encourageant la collaboration, l'automatisation et la livraison continue. Ce qui permet à l'équipe de développer et de déployer des logiciels de manière plus rapide, fiable et efficace, conformément aux principes DevOps[4].

Dans le cadre du projet actuel, comme mentionné précédemment, nous pensons qu'instaurer un mono-repos peut aider à atteindre les objectifs de modernisation fixés par l'entreprise. En premier lieu, nous croyons que cela

[4] Les cinq principes DevOps: Collaboration; Automatisation; Amélioration continue; Action centrée sur le client; Lors de la création, il tenir compte de la finalité.

favorisera une homogénéité du code, car l'utilisation de mono-repos offre une vue d'ensemble plus claire du système.

Avoir tout le code au même endroit permet de répondre plus rapidement à des questions telles que "combien de fois tel ou tel composant est-il utilisé ? », plutôt qu'à chercher la réponse dans des repository différents, sachant il possible que pas tous les repos vont être accessible à un développeur. De plus, il est possible de modifier un composant dans plusieurs packages en un seul commit. Il est également plus facile de repérer le code "mort", car nous avons accès à l'ensemble du code.

Aussi, il ne faut pas oublier que le mono-repos aide à uniformiser la façon de travailler des équipes et des individus, ce qui est essentiel pour avoir un code autodocumenté. Car, comment nous l'avons souligné plutôt, il est nécessaire d'instaurer les normes/conventions commune de travail.

Donc lors de l'adoption de la méthodologie DevOps, il est essentiel de définir clairement les conventions de l'entreprise et de les documenter. Cela faciliterait le processus de CI/CD ainsi que l'intégration des nouveaux collaborateurs. Aussi il faudra amener la compréhension des équipes vers une vison de la méthodologie comme d'une philosophie qui impose un changement de culture et d'organisation, qui impose une étroite collaboration entre les équipes et au sein des équipes.

Pour améliorer cette communication, il serait possible de mettre en place des réunions quotidiennes de scrum (DSM - Daily Scrum Meeting) autour de trois questions:

- Ce que j'ai fait hier
- Ce que je vais faire aujourd'hui
- Les problèmes que je rencontre

```
// Variable aléatoire
let x = Math.floor(Math.random() * 100) + 1;

// Affichage du nombre
console.log("x :", x);

// Vérification pair/impair
if (x % 2 === 0) {
    // Pair
    console.log("x est pair.");
} else {
    // Impair
    console.log("x est impair.");
}
```

Fig. 2

Le daily scrum présente de nombreux avantages, tant en termes de communication en entreprise que de bien-être des collaborateurs. Bien sur ce n'est pas un lieu des longs débats mais plus temps un lieu d'échange au cours duquel les collaborateurs vont pouvoir être orienté vers les bonnes personnes pour résoudre les problèmes rencontré, aussi c'est un lieu qui permettra de synchroniser l'équipe – soit être au courant et mettre au courant. Il existe beaucoup de littérature présentant des bonnes pratique du daily scrum, c'est pour cela que nous avons évoquer uniquement les points que nous considérons comme important:

- 15 minutes maximum.

- Tous les jours à heure fixe.
- De préférence debout devant le tableau du sprint backlog ou en partageant l'écran du sprint backlog si l'équipe travaille à distance.
- Chacun répond à 3 questions, évoqué plus haut, qui sont ramenées à l'objectif de sprint.

## Documentation et Code Autodocumenté.

```
1  const passwordHashed ;
```

Fig. 3

Avant de commencer l'explication du Pipeline, nous voudrons aborder la problématique de la documentation. Nous avons évoqué rapidement la solution plus haut, en avançant que les équipes doivent suivre les conventions communes. Ainsi dans cette partie nous voulons approfondir le sujet.

```
let intCount = 0;
let boolIsFound = false;
let strMessage = "Bonjour, monde !";
let arrNumbers = [1, 2, 3, 4, 5];

function calcSum(intArray) {
    let intSum = 0;
    for (let i = 0; i < intArray.length; i++) {
        intSum += intArray[i];
    }
    return intSum;
}

let intTotal = calcSum(arrNumbers);

console.log("La somme des nombres est :", intTotal);
```

Fig. 4

Pour créer, de façon général, une application de qualité, et surtout une application, sur laquelle des collaborateurs différents peuvent travail, il faut avoir un code lisible – c'est à dire compréhensible et maintenable par tous les développeurs qui pourront intervenir sur le projet. Et donc pour cela qu'il faut instaurer des conventions/règles d'écriture communes et surtout les documenter. Car la documentation est importante pour la maintenabilité. Elle permet de comprendre le fonctionnement globale du code, et de savoir quelles parties du code vont être affectées par une modification.

```
authAdminRouter.post("/logout", checkToken, async (req, res) => {
    const decoded = jwt.decode(req.token!) as DecodeToken;
    const admin = await Admin.findOne({
        where: {id: decoded.id},
    });
    if (admin) {
        await TokenAdminBlackList.create({
            token: req.token,
        });
        res.json("Déconnecté");
    } else {
        res.status(404).json("Admin nest pas trouvé");
    }
});
```

Fig. 5

Afin d'avoir un code « autodocumenté », il faut suivre les conventions de nommage. Car le but est de ne pas simplement rentre le code lisible mais compréhensible et maintenable, comme il était dit précédemment. Dans la suite de l'argumentation, vous trouverez

des illustrations issue des projets fictifs, elles aideront à illustrer les propos qui suivent.

Nous mettons accent sur la lisibilité du code, puisque étant rempli de noms d'éléments : variables, fonctions et etc. Chaque élément du code représente quelque chose en particulier. Ce qui fait que le nommage peu claire va rendre le code moins lisible et surtout moins compréhensible. Aussi, la présence, d'un nombre important des commentaire, va polluer la lecture et parfois ces commentaires vont être inutile. De ce fait pour rendre la lecture aisé, il faut appliquer des règles de nommage simple. La figure 2 illustre bien un code pollué par des commentaire inutile et nommage de variable peu transparent, contrairement au code de la figure 4, dans lequel dès le début de la lecture des variable nous pouvons comprendre à quoi se réfèrent les variable, c'est ce que nous considérons comme un code autodocumenté.

Mais pour que tout le monde appréhende cette façon d'écrire, il faut que les équipes comprennent que le nommage des classes et fonctions, devrait permettre, à un développeur extérieur, de comprendre ce qu'elle font, et ce, rien qu'en lisant son nom. Donc il faut adopter les noms qui « parlent », (voir fig. 3). Au risque de répéter les propos, nous insistant sur le fait que le noms devraient être lisibles et compréhensibles. Et pour cela il faut garder une règle en tête, suivante, *si le nom nécessite un commentaire, il faut le changer, car un nom doit exprimer une « intention ».*

Après nos différentes lecture nous nous sommes résout à proposer la notation « hongroise », dans un premier temps, car il y a plusieurs littérature qui évoque cet notation, ce qui donne à l'équipe la passibilité de retrouver les sources assez facilement, également il sera possible de retrouver certains articles, à partir des quels l'exemple (fig. 4) était élaboré, dans la bibliographie. Cependant il faut noter que nous proposions une adaptation à nous de cette notation et non celle proposé initialement par Charles Simonyi.

Dans un second temps cette convention de nommage est base sur « Camel case[5] », d'après ce que nous avons pu comprendre, est assez répondu dans la communautés de développeurs JavaScript et C#, (voir fig. 3), ce qui fait que les développeurs habitué ne vont pas trop être perturbé par l'imposition des nommages. La notation « hongroise » ajoute quelques règles à la convention « Camel case ». Cette convention de nommage oblige d'avoir les noms de variables avec des préfixes. Toutefois si ce type de notation ne convient pas, il peut servir de base à des conventions de nommage. L'exemple est illustré par la figure 4. Par la figure 5 nous avons souhaité montrer l'importance du formatage du code pour améliorer la lisibilité du code, la figure 5 montre comment faire formatage horizontal et vertical.

Pour terminer cette partie nous avons aborder rapidement l'outil que nous proposant, en tant que générateur de documentation. Pour rappeler, un générateur de documentation est un outil qui crée de la documentation destinée aux développeur. Et donc pour cette tâche peu amusante, nous avons choisi de proposer 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 par l'entreprise. Aussi il faudra faire la documentation des API, pour cela nous pensons que, une solution rudimentaire sera assez pratique, nous proposant écrire la documentation, directement dans l'éditeur de code avec le format markdown.

[5] "Camel case" veut que chaque mot commence par une majuscule sauf le premier qui commence par une minuscule

## Pipeline du Projet.

Notre proposition sous la forme d'un diagramme de pipeline, est exposé dans la figure 1, et une explication détaillée du diagramme et propositions des outils envisageables, sont dans les paragraphes suivants.

Avant tout commit et push sur la branche "dev", chaque développeur est tenu de suivre un processus de "pré-commit check". Ce processus inclut plusieurs étapes visant à assurer la qualité du code avant son intégration dans la branche principale du projet :

1. **Analyse de code statique, linting :** Cette étape consiste à utiliser des outils de linting pour détecter les erreurs de syntaxe, les violations de conventions de codage et les problèmes de style. À cette étape, nous recommandons l'utilisation d'ESLint et de Prettier, étant donné que TSlint n'est plus pris en charge par "Palantir" (source : https://www.infoq.com/fr/news/2019/02/tslint-deprecated-eslint/). Ainsi, la solution alternative directe est ESLint, tandis que Prettier représente un outil complémentaire. Toutefois, il est essentiel que l'équipe définisse les règles d'utilisation pour ces deux outils.

   Pourquoi ESLint?

   - *ESLint permet de définir des règles de style et de qualité de code, ce qui garantit une cohérence dans tout le codebase, même dans les projets impliquant plusieurs développeurs.*
   - *ESLint est très configurable, ce qui signifie que vous pouvez définir vos propres règles de linting en fonction des besoins spécifiques de votre projet ou de votre équipe.*
   - *ESLint peut être intégré dans les pipelines de build et les workflows d'intégration continue (CI) pour automatiser la vérification du code à chaque modification, garantissant ainsi la qualité du code tout au long du processus de développement.*

   *Pourquoi Prettier?*

   - *Prettier effectue le formatage automatique en se basant sur les règles ESLint, c'est un « code formatter ».*
   - *Prettier supporte plusieurs langages comme TypeScript, ES2017 etc...*

2. **Tests unitaires :** Le développeur lance les tests unitaires localement pour s'assurer du bon fonctionnement de chaque composant du code, garantissant ainsi son fonctionnement isolé. Pour cette étape, nous proposant d'utiliser Jest, c'est un framework de test JavaScript maintenu par une communauté de contributeurs open source et d'employés de Facebook.


Fig. 6

*« La pyramide des tests est une excellente métaphore visuelle qui décrit les différentes couches de test et la quantité de tests à effectuer... »* Haaker, W. (2021). *Comment la pyramide d'automatisation des tests est-elle utilisée dans le développement de logiciels? parasoft.com*

Pourquoi Jest?

- *Jest est livré avec une configuration par défaut robuste qui couvre la plupart des cas d'utilisation courants. Il offre également une grande flexibilité pour personnaliser la configuration selon les besoins spécifiques du projet.*
- *Jest est compatible avec TypeScript et peut être configuré pour compiler automatiquement le code TypeScript avant d'exécuter les tests.*
- *Étant largement adopté, cet outil bénéficie d'une documentation abondante et de nombreux tutoriels, facilitant ainsi sa maîtrise par les développeurs grâce aux ressources disponibles en ligne.*

3. **Build en local :** Après la validation des tests unitaires et de l'analyse de code, le développeur procède à un build en local pour vérifier la compilation sans erreur du code. À ce stade, l'utilisation de TypeScript Compiler (tsc) est essentielle, bien que le recours à des alternatives puisse être envisagé, mais nous ne pensons pas que cela est très utile de changer (tac) pour un autre outil, car:

    - *TypeScript Compiler est l'outil officiel de compilation fourni par Microsoft. Il est régulièrement mis à jour et maintenu, ce qui garantit un support continu et des améliorations de performances.*
    - *Étant l'outil officiel, TypeScript Compiler est conçu pour être entièrement conforme aux normes et aux spécifications de TypeScript. Cela garantit une compatibilité maximale avec le langage et ses fonctionnalités.*
    - *De nombreux projets TypeScript sont configurés pour utiliser TypeScript Compiler en raison de sa longue histoire d'utilisation et de sa fiabilité. Utiliser (tsc) assure une cohérence avec les standards de l'écosystème TypeScript.*

Une fois les vérifications préalables au commit effectuées avec succès, le développeur est autorisé à pousser ses modifications vers la branche "dev". Avant toute fusion dans la branche "dev", les modifications doivent être soumises à une validation de l'équipe(Product Owner,  Scrum Master, Lead Développeur, QA). Chaque membre de l'équipe apporte son expertise pour garantir que les modifications apportées au code sont examinées de manière approfondie. Aussi, mise en place de la culture DevOps implique la collaboration et la responsabilité partagée pour assurer une livraison de logiciels réussie et fiable.

Suite à la validation du pull request, le pipeline CI/CD prend le relais pour automatiser le processus de déploiement, à l'aide de GitHub Action, car son fonctionnement est très pratique, du fait que les workflows GitHub Actions sont déclenchés par des événements spécifiques, tels que des push de code, des créations de pull requests, des créations de tags, etc. Cela permet de déclencher automatiquement des actions en réponse à des événements spécifiques dans un référentiel GitHub.

1. **Création de l'image Docker :** Nous proposons d'assurée cet étape par Git Actions, qui compile le code et génère une image Docker contenant l'application.

2. **<u>Tests unitaires et tests end-to-end (E2E) :</u>** Les tests sont exécutés sur l'image Docker afin de vérifier le bon fonctionnement de toutes les fonctionnalités de l'application, avec un contrôle du code coverage pour assurer une couverture suffisante des tests, généralement autour de 80 à 90%. Nous estimons que conserver Jest pour les tests unitaires est préférable, car l'introduction de multiples outils différents pourrait compliquer la gestion pour les développeurs. Cette approche évite la nécessité de jongler avec plusieurs frameworks simultanément, compte tenu du nombre déjà conséquent d'outils utilisés dans les projets de l'entreprise. Pour les tests end-to-end, nous avons opté pour Playwright. Ce choix est guide par la lecture du témoignage de Diaz, L. (2023). *Why I switched from Cypress to Playwright*. Medium.com

    1. **<u>Validation du build :</u>** Un autre build est déclenché pour s'assurer que l'application fonctionne correcte ment dans un environnement de développement.

Une fois que tous les tests ont été réussis, une demande de fusion (pull request) est créée afin d'intégrer les modifications dans la branche principale du code source. Le processus est similaire à celui de la branche "dev", à une différence près : nous proposons d'effectuer des tests d'intégration à la place des tests unitaires. Les tests d'intégration évaluent le système dans son ensemble, reproduisant au plus près son fonctionnement réel. Ils incluent la vérification des composants externes tels que les bases de données, les services web ou les systèmes de fichiers, permettant ainsi de contrôler la communication entre l'application et ces éléments périphériques. Pour ces tests, nous recommandons l'utilisation de Jest ainsi que de SuperTest. SuperTest, extension de la bibliothèque SuperAgent, est une bibliothèque de tests HTTP pour Node.js, idéale pour testr des API REST ou des services HTTP dans des applications.

```
{
  error: 0,
  warn: 1,
  info: 2,
  http: 3,
  verbose: 4,
  debug: 5,
  silly: 6
}
```

Fig. 7

En complément, l'utilisation d'une bibliothèque d'assertions comme Chai est recommandée. Chai offre une syntaxe expressive et prend en charge différents styles d'assertions tels que "expect", "should", "assert". De plus, Chai s'intègre parfaitement avec SuperTest.

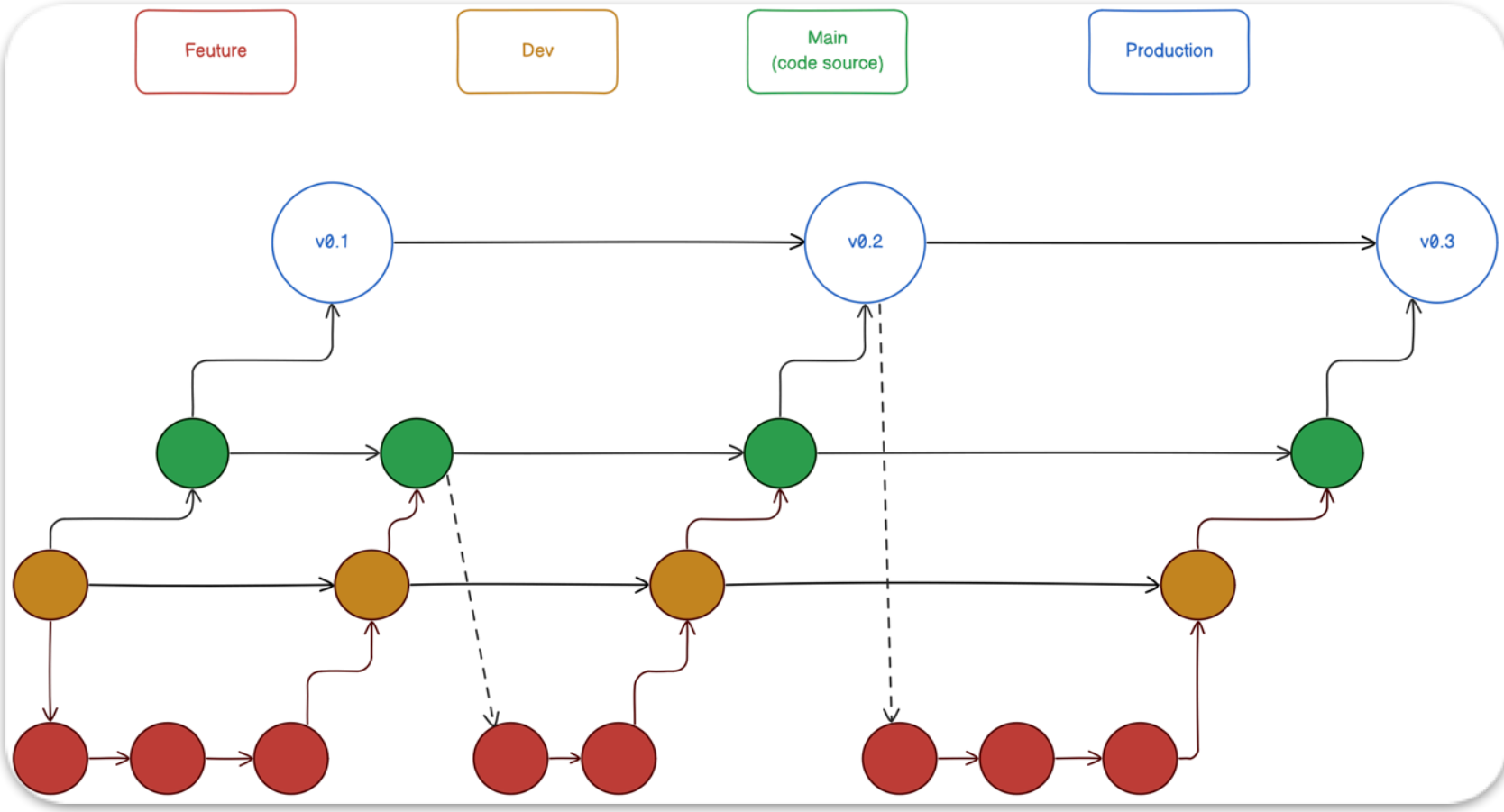
Une fois les tests d'intégration et de linting réussis, une livraison automatique est effectuée vers un registre Docker Hub. Cette automatisation permet au projet d'être plus facilement vérifié par l'équipe d'Assurance Qualité (QA), car plusieurs batteries de tests ont été effectuées, allégeant ainsi leur charge de travail.

Une fois la vérification effectuée, l'application peut être déployée sur l'environnement de staging[6]. Cet environnement peut être utilisé pour effectuer des démonstrations et des validations avec les clients ou les parties prenantes, en plus nous préconisons utilisation d'un outils s'intitulant « Winston ». Cela permettra de visualiser les nouvelles fonctionnalités et les changements proposés avant le déploiement en production, mais aussi, en cas des

[6]*Le terme staging environment (en français : environnement de simulation) désigne un environnement serveur permettant de faire des tests dans des conditions les plus proches de la réalité. Un environnement de simulation se compose d'un matériel semblable à la variante de production. Pendant cette phase, on accède à des sections de la base de données et il est possible qu'un certain nombre d'utilisateurs disposent d'un accès afin de testr les fonctions. Source: Staging Environment – Ryte Wiki - Wiki du marketing digital*. (s. d.). https://fr.ryte.com/wiki/Staging_environment

dysfonctionnement rencontré, avoir une journalisation[7]. Grace à Winston, qui est une bibliothèque de journalisation de Node.Js. L'implémentation de la journalisation au staging et à la production répondra à la problématique de traçabilité des erreurs. Et justement, Winston, à nos yeux, est un outil parfait pour tracer les erreurs et voir où est ce que cette erreur se produit dans le code.

PlayerZero (2023) propose:




Git Flow (Fig. 8)

*Ce diagramme est visible à l'adresse: https://app.eraser.io/workspace/5hBSG2Rpn8HSUqyrgAqg?origin=share*

*« Chaque fois que quelque chose ne va pas dans une application, les développeurs ont la possibilité de retrouver sa source via une trace de pile. Mais... se fier uniquement aux traces de la pile ne peint qu'une demi-image. Le traçage vous fournira des informations sur l'exécution d'une application, y compris le flux de requêtes, les requêtes de base de données, les appels d'API externes et les mesures de performances - ce qui peut être très utile. En général, les traces sont utilisées pour comprendre le comportement d'une application à un niveau granulaire et pour identifier les goulots d'étranglement des performances.*

*La journalisation remplit l'autre moitié de l'image. La journalisation est le processus d'enregistrement des messages et des événements liés à une application, tels que les erreurs, les avertissements, les mesures de performances et les informations de débogage. La connexion à Node.js est utilisée pour diagnostiquer les problèmes, comprendre le comportement des utilisateurs et améliorer les performances et la stabilité globales d'une application. Il fournit également un enregistrement de l'activité dans une application, permettant aux développeurs d'identifier et de déboguer les problèmes et de prendre des décisions plus éclairées. »*

---

[7] *La journalisation(logging en anglais) consiste à historiser les événements (connexion, erreurs, exceptions) survenus dans une application. Source: Sécurisez le traitement des erreurs et des logs*. (s. d.). OpenClassrooms. https://openclassrooms.com/fr/courses/1761931-securisez-vos-applications/5702858-securisez-le-traitement-des-erreurs-et-des-logs

Par conséquent, Winston (d'après PlayerZero encore une fois), est une bibliothèque très populaire pour Node.js, mais aussi elle est très bien documenté, ce qui aidera l'intégration de l'utilisation de cet outil au sein de l'équipe. Pour argumenter ce chois, nous avons lu plusieurs articles ce qui nous a conduit à constater que Winston, est un outil assez flexible et il permet de configurer les formats de message et les destinations de stockage et les « niveaux » des message(fig. 7). Ce qui va beaucoup aider, les développeur, de comprendre les loggues et donc débogguer efficacement.

Ainsi notre pipeline, à nos yeux, assurera une qualité et fiabilité du logiciel. Notre propositions des tests, était inspirait par l'illustration (et l'article en lui-même bien sûr) de la pyramide d'automatisation des tests, que nous avons vu dans l'article s'intitulant (fig. 6)« Comment la pyramide d'automatisation des tests est-elle utilisée dans le développement de logiciels? ». La pipeline était inspiré par plusieurs videos de témoignage que nous avons pu visionner, il est possible de les retrouver dans la bibliographie comme toute la littérature sur laquelle nous avons pu s'appuyer our élaborer ce dossier. Pour terminer cette partie nous souhaitons, aborder rapidement, le Git Flow que nous proposons pour le projet, pour être exhaustif et surtout le plus sommaire possible, nous allons présenter une illustration et aborderons les points les plus important.

Sur cette représentation (figure 8) nous pouvons voir comment nous concevons le Git Flow du projet. Nous voulions montrer comment nous voyons le travail avec les branche. Comme il est possible de constater il y a des branches permanentes comme « Dev », « Main » et « Production », et il y a les « Feuture » pour designer le travail de chaque développeur, alors que « Dev » et « Main » sont les fruits du regroupement des tous les « Feuture ». Aussi il possible de voir, que nous avons prévu, le cas éventuel du problème en « staiging » ou en production, cela est illustré par les flaches qui nous envoi directement vers les « Feuture ». Quand nous évoquons des problèmes éventuels, nous faisons références soit aux soucis lie à l'intégration des nouvelles fonctionnalité mais aussi au fait que les clients peuvent faire un retour suite au staging et soit refuser, soit vouloir ajouter des nouvelles fonctionnalités. Nous pensons que notre approche, évitera des problèmes de régressions, mais cela oblige d'avoir deux branches qui sont pratiquement les mêmes, mais portent deux noms différents « Main » « Production », sauf que la branche « Production » peut recevoir du code une fois que, staging est validé.

# Bilan de l'étude.

Pour conclure cette étude, nous voudrons revenir sur les propositions données, pour atteindre les objectifs. Aussi nous souhaitons faire un point sur les solutions que nous avons proposé et ne pas proposés.

## Récapitulatif.

Dans le cadre de cette étude, nous avions comme objectifs imposés, rassembler tous dans un seul repository, améliorer le workflow de développement, la qualité de code et faire en sorte que par la suite le code soit plus homogène. Egalement, il fallait éviter les régression et avoir une documentation à jour. Autrement parlant notre défit était d'arriver vers une modernisation de la méthodologie du travail des équipes.

Donc pour amener l'entreprise vers cette objectif, dans un premier temps nous avons évoqué l'importance de la définitions des objectifs que nous qualifions de « matériel » et « mercantile », par là on sous-entend tout ce qui est en rapport avec KPI, SLA, SLI et autres.  Même si, ces données ne sont pas toujours intéressant pour les développeur, il sont assez important pour l'avenir de la relation entre l'entreprise et un client donnée.

Ainsi avoir une vision claire des objectifs permettra à l'entreprise de mieux organiser le travail et transmettre les besoins des clients aux équipe. Egalement cette définition va aider mieux saisir l'impacte, de la méthodologie DevOps, sur la productivité. Car *DevOps a pour objectif de faciliter la production d'applications et de services de qualité à un rythme élevé afin d'apporter de la valeur aux clients* (Pourquoi la Culture DevOps Est-elle Bénéfique Pour Votre Entreprise ? , s. d.). Bien sûr ce n'est pas le seul objectif de cette philosophie, mais cela ajoute une valeur pour l'entreprise.

Pour améliorer le workflow, la qualité de code, et l'écriture de la documentation, nous avons proposé un certain nombre d'outils et conventions, qui nous sont semblé appropriés pour cette entreprise et ses projets, tout en prenant en compte les critères d'une approche DevOps, dans le travail. Nous avons proposer utiliser les tests statiques avant les commits pour rendre le code « propre » au moment de son ajout aux branches de travail. De la même façon, Git Actions pour automatisation des tests unitaire, e2e et d'intégration après les commits et les pull request, afin de faciliter le travail de QA, en lui permettant de faire des tests physique et écrire la documentation comme c'était évoqué dans la partie portant sur les problématiques. Egalement le pipeline proposé va aide d'avoir un code fonctionnel à chaque stade de développement. De plus l'utilisation d'un outil de logging permettra avoir un retour, structuré, sur le fonctionnement de code et donc possibilité de correction.

## Discussion.

Au cours de nos recherches pour ce projet, nous avons examiné plusieurs témoignages portant sur les pratiques des personnes et équipes. Cette exploration nous a conduit à remettre en question certaines propositions que nous avions envisagées mais finalement rejetées. Par exemple, nous avons envisagé l'utilisation du "Husky Pre-commit Hook", bien que cela puisse sembler intéressant. Notre décision de ne pas intégrer cet outil dans notre approche, repose sur notre conviction qu'il est préférable que les développeurs puissent inspecter le code avant son ajout au référentiel, et donc utiliser ESLint pour corriger le code en direct.

Nous avons également évoqué l'utilisation de Prettier, sachant qu'il peut être complémentaire à ESLint. Cependant, Prettier présente un risque potentiel car il fonctionne de manière indépendante avec son propre parser et ses propres règles, ce qui peut entraîner des conflits potentiels. Heureusement, dans l'écosystème JavaScript, ESLint et Prettier peuvent coopérer (https://prettier.io/docs/en/eslint.html). ESLint peut ainsi déléguer à Prettier uniquement la tâche de formatage du code, en laissant ESLint se concentrer sur d'autres aspects de l'analyse. Pour que cette collaboration fonctionne, une configuration commune entre ces outils est nécessaire, comme mentionné précédemment. Toutefois, ce choix est discutable et il mérite d'être aborder au cours de début de la mise en place de la méthodologie, également il faudra reborder les autres propositions portant sur le nommage et les outils de la documentation. Car la philosophie DevOps est porté par l'envie de faire collaborer les équipes.

Egalement, nous avons proposé d'utiliser Playwright à la place de Cypress, mais encore une fois cela est discutable car depuis un certain temps Cypress permet de testr les API. Mais pour l'instant notre choix être porté sur les outils qui sont davantage familier pour nous, également ce choix était motivé par l'article de Dmitrii Bormotov (2023) « Playwright vs Cypress pour les tests automatisés de l'API REST : qui arrive en tête ? », qui préconise l'utilisation de Playwright pour plusieurs raisons, comme par exemple il n'est fourni avec aucun navigateur prêt à l'emploi - vous devez les installer avec une commande distincte, ce qui donne la liberté d'installer les navigateur nécessaire.

La demande de de l'entreprise était de rassembler tous les projet dans un seul référentiel. Effectivement c'est une bonne idée pour, faciliter le travail et ne pas avoir de problèmes au moment des modifications des composants. Comme toute approche, cette approche a ses propres inconvénients. L'inconvénient qui nous a sauté aux yeux et celui des autorisation. Autrement écrit, quand nous mettons des permission de lecture et écriture, nous donnons la permission pour le repository entier. Bien sûr il y a des solutions pour palier à ce problème, une des solutions nous avons trouvé à l'adresse suivant: https://github.com/smulikHakipod/git-subfolder-permssions .

Donc pour l'instant c'est une solution approprié mais il ne faut pas oublier qu'en mettant en place un mono repos, il y aura des défis à soulever. Un des problème, c'est Git va suivre tout l'état de l'arbre et donc il y aura probablement des éventuel problème de performance, quand les développeurs vont faire certaines actions avec git, comme par exemple « git status ». C'est pour cela nous pensons qu'il faudra revoir cette solution.

Nous avons proposé, en plus de pipeline, un git flow avec utilisation des certaines branches pour palier le problèmes de régressions, toutefois nous pensons que les deux pourrons être vraiment considéré comme une solutions stable une fois ils sont vérifiés dans la vie réelle. Car nos propositions restent peut être un peu loin de la réalité du fais que chaque projet a besoin ses propres méthodes et il ne faut pas avoir peur de changer des choses, apporter une vison nouvelle, l'importance que les équipes aient une communication saine et qu'elles soient toujours prête a discuter et remettre en question la façon de travailler.

# Conclusion.

Pour conclure ce dossier, et utiliser « je » pour la première fois à la place de « nous » de modestie, que vous avez pu lire à plusieurs reprise, je voudrais aborder le « en-dehors » du sujet. Les recherches sur cet étude de cas, m'ont permis de voir le rôle réel de la méthodologie, dont j'ai vu en cours, par les biais des témoignages que j'ai lu. Egalement ce dossier m'a permis de me rendre compte de la multitude d'outils différents qui ont ses propres points fort et points faible. Pour chaque situation on peut trouver un outil qui répondre mieux au besoin.

Il est certain les outils abordés dans ce dossier, peut être, ne sont pas les meilleurs pour répondre au besoin de l'entreprise car il est, selon mon avis, comprendre et appréhender tous les outils existante, car à chaque fois quand j'essayais de comparer des outils par le biais de la lecture, chaque nouvel article traité de plusieurs outils que je ne connaissait pas ce qui fait le travail de la comparaison se transforme en travail de veille, mais ayant le temps limité pour l'écriture de l'étude j'ai du me résoudre à proposer des outils que nous avons abordé en cours.

Egalement, je suis conscient que je n'ai pas pu aborder tous les points à l'écrit, ou les aborder en profondeur, car si je l'aurais fait le dossier aurait fait une dizaine de pages en plus. Donc je préférais les laisser pour revenir sur eux à l'oral. Comme par exemple, le Docker était mentionné mais abordé en détail, ou Winston était également présente assez sommairement. Les tests e2e n'étais pas abordé non plus.

# Bibliographie.


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

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

Atlassian. (s. d.-b). *SLA, SLO et SLI : les différences | Atlassian*. https://www.atlassian.com/fr/incident-management/kpis/sla-vs-slo-vs-sli

Atlassian. (s. d.-c). *SLA : Tout ce que vous devez savoir | Atlassian*. https://www.atlassian.com/fr/itsm/service-request-management/slas

Benbrahim, R. (2022, août 31). ESLINT : Comment coder proprement en JavaScript - WeLoveDevs.com. WeLoveDevs.com. https://welovedevs.com/fr/articles/eslint/

*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

Chaitanya, K. (2021, 13 décembre). Rest API testing with Mocha, Chai, SuperTest - Kishan Chaitanya - Medium. *Medium*. https://kishanchaitanya.medium.com/api-testing-using-mocha-chai-and-supertest-a7c7edc96c24

Chazelle, J. (2020, 14 juillet). *Utiliser ESLINT et Prettier pour un code de qualité*. Jérémie Chazelle. https://jeremiechazelle.dev/utiliser-eslint-et-prettier-sous-visual-studio-code-webstorm-phpstorm-pour-un-code-de-qualite/

Code documentation for JavaScript with JSDOC : An Introduction. (2020, 2 février). https://www.valentinog.com/blog/jsdoc/

Contributeurs aux projets Wikimedia. (2023, août 2). *JSDOC*. Wikipédia. https://fr.wikipedia.org/wiki/JSDoc

*Cypress, Tests de bout en bout (E2E) | Fullwave*. (s. d.). Fullwave. https://fullwaveagency.com/blog/cypress-tests-de-bout-en-bout-e2e/

Darras, G. (2019, 17 septembre). *Documenter son projet avec Storybook*. Blog Ineat. https://blog.ineat-group.com/2019/09/documenter-son-projet-avec-storybook/

De Best, B. (2022, 31 mai). *Plan DevOps - SLA et exigences non fonctionnelles*. Welcome IT Professional. https://fr.itpedia.nl/2017/07/14/devops-plan-slas-and-non-functional-requirements/

Delattre, L. (2022, 7 septembre). *DevOps : de la philosophie aux outils*. IT For Business. https://www.itforbusiness.fr/devops-de-la-philosophie-aux-outils-40753

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

Diaz, L. (2023, 7 novembre). Why I switched from Cypress to Playwright - Lucy Diaz - medium. *Medium*. https://medium.com/@oldiazg/why-i-switched-from-cypress-to-playwright-dc41ce4d5e1b

*Documentation - JSDOC reference*. (s. d.). https://www.typescriptlang.org/docs/handbook/jsdoc-supported-types.html

Doglio, F. (2023, 12 avril). Documenting your TypeScript projects : There are options. *Medium*. https://blog.bitsrc.io/documenting-your-typescript-projects-there-are-options-da7c8c4ec554

Driat, L. (2021, 6 janvier). *Écrire des commentaires de code PARFAITS : le guide ultime*. IT Expert. https://itexpert.fr/blog/commentaires-parfaits/

*Écrivez du code autodocumenté*. (s. d.). OpenClassrooms. https://openclassrooms.com/fr/courses/6398056-ecrivez-la-documentation-technique-de-votre-projet/6904236-ecrivez-du-code-autodocumente

Frontend Channel. (2021, août 19). **Ликбез по** CI/CD **для** frontend'**а на примере** GitLab / **Тимофей Тиунов** [Vidéo]. YouTube. https://www.youtube.com/watch?v=BIY_J0Ba4Cc

*Gérer les assertions avec CHAi – formation tests en JavaScript*. (s. d.). Grafikart.fr. https://grafikart.fr/tutoriels/assertions-chai-654

Goffinet, F. (2020, août 9). *Introduction à l'Infrastructure as Code*. cisco.goffinet.org. https://cisco.goffinet.org/ccna/automation-programmabilite-reseau/infrastructure-as-code/

Haaker, W. (2024, 25 janvier). Comment les pyramides d'automatisation des tests sont-elles utilisées dans le développement de logiciels ? Parasoft. https://fr.parasoft.com/blog/testing-automation-pyramids-for-software-development/

Isaiah, A. (2023, 25 octobre). *A complete guide to Winston logging in Node.js*. Better Stack Community. https://betterstack.com/community/guides/logging/how-to-install-setup-and-use-winston-and-morgan-to-log-node-js-applications/

Jacobsen, T. (2023, 14 décembre). Mono-repository : nos développeurs réalisent une transition majeure pour collaborer plus efficacement. Infomaniak Network News. https://news.infomaniak.com/monorepo-collaborer-efficacement/

Khatri, M. F. (2023, 6 janvier). API testing using SuperTest ! - Mohammad Faisal Khatri - Medium. *Medium*. https://medium.com/@iamfaisalkhatri/api-testing-using-supertest-ea37522fa329

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/

Maniez, D. (s. d.). *La notation hongroise*. Developpez.com. https://dominiquemaniez.developpez.com/Nothongroise/

Martinez, J. (2023, 9 mars). *À la découverte des monorepos pour partager son code JS | DARVA*. DARVA. https://www.darva.com/fr/2023/03/02/a-la-decouverte-des-monorepos-pour-partager-son-code-js/

Martins, J. (2023, 26 septembre). Qu'est-ce qu'un KPI (indicateur de performance) ? [2023] • Asana. *Asana*. https://asana.com/fr/resources/key-performance-indicator-kpi

Mazzoni, L. (2023, 17 février). *Pourquoi utiliser Cypress ? - UpSkill4IT*. UpSkill4IT. https://upskill4it.com/pourquoi-utiliser-cypress/

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

Monorepo : Est-ce que ça vaut le coup ? | Maxence Poutord. (2020, 22 décembre). *Monorepo : est-ce que ça vaut le coup ? | Maxence Poutord*. https://www.maxpou.fr/monorepo-pros-and-cons-fr

*Notation hongroise : Définition et explications*. (s. d.). Techno-Science.net. https://www.techno-science.net/definition/11395.html

*NPM : Winston*. (s. d.). Npm. https://www.npmjs.com/package/winston

Orie, C. (2019, 8 novembre). *Testing NodeJS/Express API with JEST and SuperTest*. DEV Community. https://dev.to/nedsoft/testing-nodejs-express-api-with-jest-and-supertest-1km6

PlayerZero. (2023, 15 février). *Les 10 meilleures bibliothèques de journalisation Node.js*. HackerNoon. https://hackernoon.com/fr/les-10-meilleures-biblioth%C3%A8ques-de-journalisation-nodejs

*Pourquoi et comment créer des modules sur NodeJS*. (2016, 23 septembre). M@XCode. https://maximilienandile.github.io/2016/09/23/Pourquoi-et-comment-creer-un-module-avec-nodejs/

Pourquoi la culture DevOps est-elle bénéfique pour votre entreprise ? (s. d.). https://carrieres.groupeozitem.com/blog/culture-devops

*Qu'est-ce que le Daily Scrum Meeting ?* (s. d.). https://www.hubvisory.com/fr/blog/le-daily-scrum-meeting

*Réalisez vos premiers tests unitaires avec JEST*. (s. d.). OpenClassrooms. https://openclassrooms.com/fr/courses/7159306-testez-vos-applications-front-end-avec-javascript/7332796-realisez-vos-premiers-tests-unitaires-avec-jest

*Sécurisez le traitement des erreurs et des logs*. (s. d.). OpenClassrooms. https://openclassrooms.com/fr/courses/1761931-securisez-vos-applications/5702858-securisez-le-traitement-des-erreurs-et-des-logs

Solderea, I. (2024, 6 janvier). *Cypress vs Playwright : A Detailed Comparison | LambdaTest*. LambdaTest. https://www.lambdatest.com/blog/cypress-vs-playwright/

Staging Environment – Ryte Wiki - Wiki du marketing digital. (s. d.). https://fr.ryte.com/wiki/Staging_environment

Stemmler, K. (2021, 19 décembre). How to use ESLint with TypeScript | Khalil Stemmler. *khalilstemmler*. https://khalilstemmler.com/blogs/typescript/eslint-for-typescript/

*Test d'intégration : quand et pourquoi ?* (s. d.). https://www.mobiapps.fr/blog/test-dintegration-quand-et-pourquoi

Testim. (2022, 2 février). TypeScript Unit Testing 101 : A Developer's Guide. AI-driven E2E automation with code-like flexibility for your most resilient tests. https://www.testim.io/blog/typescript-unit-testing-101/

Thomas. (2021, 9 décembre). *Partage d'expérience : un tuto inspiré des méthodes de travail CentreOn pour améliorer le travail d'équipe sur REACT avec ESLINT et Prettier - Centreon*. Centreon. https://www.centreon.com/fr/partage-dexperience-un-tuto-inspire-des-methodes-de-travail-centreon-pour-ameliorer-le-travail-dequipe-sur-react-avec-eslint-et-prettier/

*Veiller à la qualité de votre code et sa documentation*. (s. d.). CNIL. https://www.cnil.fr/fr/veiller-la-qualite-de-votre-code-et-sa-documentation

*Vérifiez que votre code respecte les conventions de codage*. (s. d.). OpenClassrooms. https://openclassrooms.com/fr/courses/7697016-creez-des-pages-web-dynamiques-avec-javascript/7911268-verifiez-que-votre-code-respecte-les-conventions-de-codage

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/

webDev. (2020, 4 février). ***Просто о*** *CI/CD (****Непрерывная интеграция и доставка****)* [Vidéo]. YouTube. https://www.youtube.com/watch?v=7S1ndRRht6M

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

Yeeply. (2022, 15 novembre). *Qu&rsquo ; est-ce que le test unitaire ? comment s&rsquo ; y prendre ?* Yeeply. https://www.yeeply.com/fr/blog/test-unitaire-comment-sy-prendre/

Zerial, A. (2024, 11 janvier). Les avantages de l& # 039 ; architecture modulaire informatique. *Organisation Performante*. https://www.organisation-performante.com/les-avantages-de-larchitecture-modulaire-informatique/

**Мир IT с Антоном Павленко**. (2023, 10 janvier). ***Что такое*** *SLI, SLO, SLA* ***КАК СЧИТАТЬ, что такое ?*** [Vidéo]. YouTube. https://www.youtube.com/watch?v=14YSD5b0jHE

**Программист, T**. (2018, 7 décembre). **Сначала – монолит, или правильный путь к микросервисной архитектуре**. Tproger. https://tproger.ru/translations/monolithfirst