

Sommaire

Sommaire 2

Introduction 3

Présentation de l’entreprise 3

problématique 4

**compréhension du besoin et la traduction technique de la réponse apportée 4**

KPI, SLI et SLA dans la Méthodologie de Projet 4

la Modernisation du Workflow 6

Discussion 10

Bibliographie 10

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 systèmes logiciels, tout en favorisant une culture d'innovation et 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 encourager la collaboration, optimiser les processus de développement et de déploiement, réduire les risques et favoriser l'agilité face au changement.

À travers cette étude, nous proposerons des recommandations et des stratégies pour une mise en œuvre efficace de l'approche DevOps, adaptée aux objectifs et aux besoins uniques de l'entreprise. Cela facilitera son développement continu et son succès dans un environnement concurrentiel en perpétuelle évolution.

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 l'accès web et 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 testes, mais le QA ne réalise que des testes physiques et n'a pas le temps d'effectuer des testes automatisés. Sur le back-end et l'électronique, des testes 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.

compréhension du besoin et la traduction technique de la réponse apportée

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 pour le succès des projets, tant dans leur gestion que dans leur exécution. Dans ce contexte, la définition précise des indicateurs clés de performance (KPI[[1]](#footnote-1)), des objectifs de niveau de service (SLI[[2]](#footnote-2)) et des accords de niveau de service (SLA[[3]](#footnote-3)) revêt une importance cruciale. Ces éléments offrent une feuille de route permettant de mesurer, d'évaluer et de garantir la qualité, la fiabilité et la performance des produits ou services développés.

Dans le cadre du projet, axé sur la création d'outils de diagnostic dans le contrôle qualité, l'importance des KPI, SLI et SLA est particulièrement accentuée. Car la société, reconnue pour son expertise dans la mesure de la qualité de l'air et d'autres domaines connexes, s'engage à fournir des solutions de haute qualité à ses clients. Ainsi que, 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 cruciales pour garantir la qualité, la fiabilité et la performance des services fournis.

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 différentes mesures récupérés 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.

* 1. 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.

* 1. Temps de Résolution des Incidents :

Spécifie le délai maximal acceptable pour résoudre les incidents ou les problèmes signalés.

En prenant en compte ces indicateur nous pourrons soit calculer un KPI (dans le cas ou on choisi une donnée pour SLI et une donnée pour SLA), soit deux. La pertinence de choix doit être discuter en équipe, afin de instaurer une vision général des attentes au sein de l’équipe.

La formule basique pour calculer le KPI pourrait être :

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

Pour amorcer le processus de modernisation, notre proposition préliminaire vise à regrouper tous les projets au sein d'un seul référentiel, 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, une pipeline CI/CD rigoureuse et l'adoption de normes communes, par conséquent ceci favorise 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]](#footnote-4).

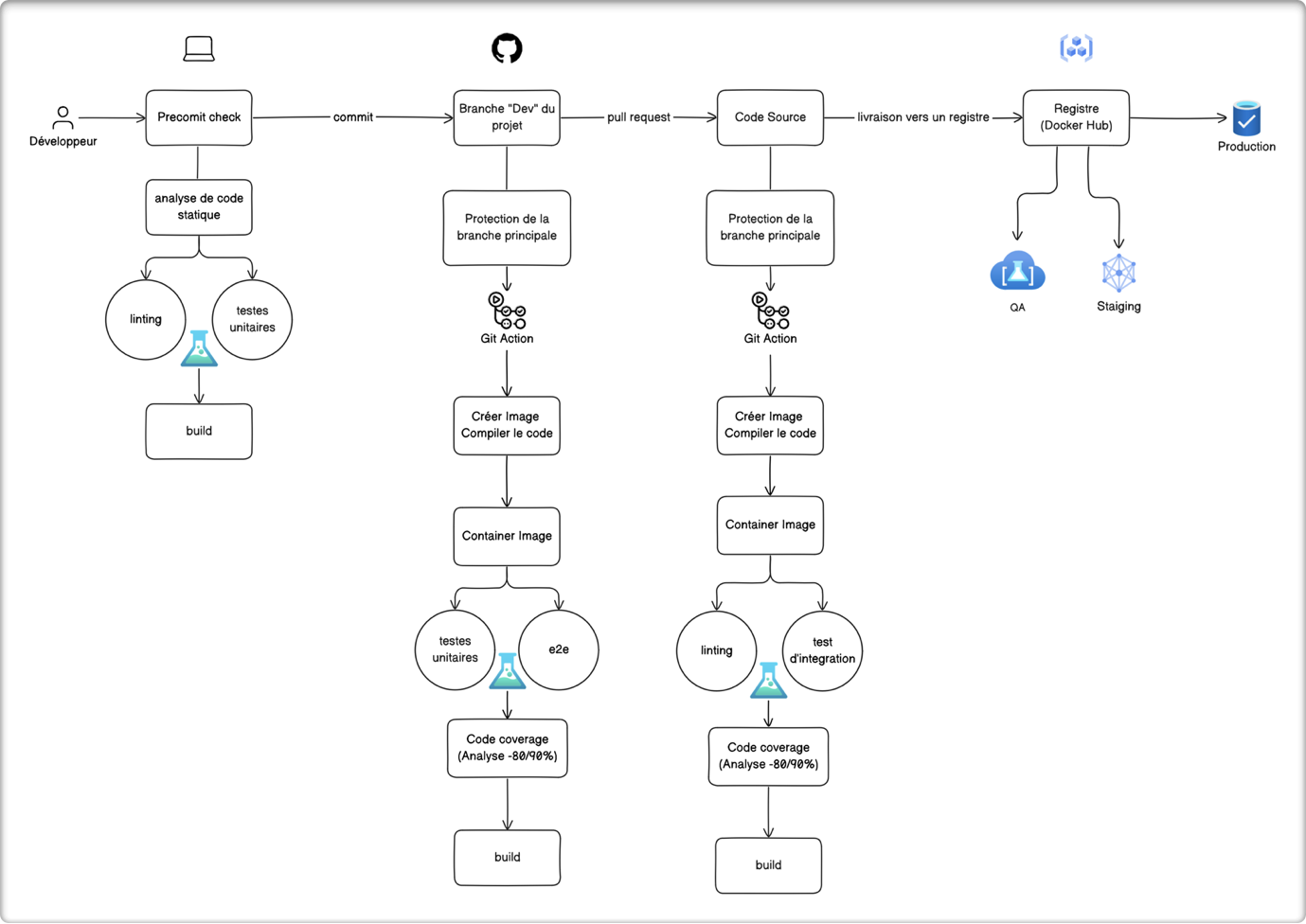


Figure 1.

Cette représentation vous pouvez retrouver dans mon « workspace » sur https://app.eraser.io/workspace/eUuXTC7HOgWm6g2IIzZq?origin=share

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…*

* 1. Testes unitaires : Le développeur lance les testes 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 teste JavaScript maintenu par une communauté de contributeurs open source et d'employés de Facebook.

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 testes.*

*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.*

* 1. Build en local : Après la validation des testes 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.

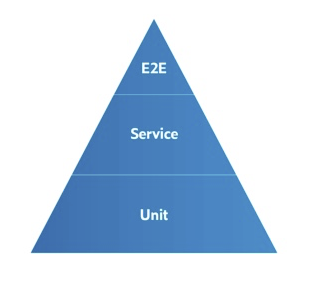


Figure. 2

« 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

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.

1. Testes unitaires et testes end-to-end (E2E) : 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. Validation du build : Un autre build est déclenché pour s'assurer que l'application fonctionne correctement 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 "des", 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 tester des API REST ou des services HTTP dans des applications.

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 à l'application d'être plus facilement vérifiée 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. Cet environnement peut être utilisé pour effectuer des démonstrations et des validations avec les clients ou les parties prenantes, leur permettant ainsi de visualiser les nouvelles fonctionnalités ou les changements proposés avant le déploiement en production.

Ce pipeline assure une approche cohérente et automatisée pour garantir la qualité et la fiabilité du logiciel, conforme à notre vision professionnelle du développement logiciel. Notre propositions des testes, était inspirait par la pyramide d'automatisation des testes (fig. 2), que nous avons vu dans un article s’intitulant « Comment la pyramide d'automatisation des tests est-elle utilisée dans le développement de logiciels? », (l’article en lui-même bien sûr).

Discussion

Au cours de nos recherches pour ce projet scientifique, nous avons examiné plusieurs témoignages concernant les pratiques de diverses 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 de la problématique 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 analyseur 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 formater le 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.

Nous avons pas utilisé Cypress car à la place on a pris playwright

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/

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

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/

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

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

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/

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/

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

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

*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/

*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

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/

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/

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/

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. Dans cette mise à jour, nous verrons comment définir et suivre des KPI adaptés à votre activité. [↑](#footnote-ref-1)
2. Les SLI sont les indicateurs factuels de la disponibilité de votre service. Ils représentent l’état de votre 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. [↑](#footnote-ref-2)
3. SLA est un engagement entre deux parties. Cela peut être entre deux entreprises (un provider et un client) ou bien entre départements dans une grande entreprise. Le SLA va définir votre engagement sur la disponibilité du service. [↑](#footnote-ref-3)
4. Les cinq principes DevOps: Collaboration; Automatisation; Amélioration continue; Action centrée sur le client; Lors de la création, il tenir compte de la finalité. [↑](#footnote-ref-4)