Introduction

Dans le cadre de ce projet, notre objectif est de construire une solution robuste et moderne qui intègre ce qui se fait de meilleurs. Pour y répondre, le choix des technologies détermine directement la performance, la maintenabilité et la facilité d'évolution de l'application.

Chaque décision technologique a été guidée par plusieurs critères :

1. **Adéquation avec les besoins du projet :** Les technologies choisies doivent répondre précisément aux objectifs fixés, comme l'automatisation des déploiements ou la gestion efficace des environnements.
2. **Fiabilité et scalabilité :** Les outils sélectionnés doivent garantir la stabilité des systèmes et s'adapter à des charges de travail croissantes.
3. **Pratiques modernes :** Enfin, il était crucial d'adopter des solutions alignées avec les standards actuels du développement et de pouvoir expérimenter de nouvelles technologies.

Dans ce document, nous allons détailler les choix technologiques réalisés, expliquer pourquoi ces outils ont été retenus et les comparer à d'autres options pour démontrer leur pertinence.

Infrastructure DevOps

Outil : Docker

Documentation : https://docs.docker.com/

Docker est un outil qui permet de créer des environnements isolés pour exécuter les applications. L'avantage de Docker est qu'il garantit que l'application fonctionne de la même manière sur toutes les machines. De plus, il est largement adopté et possède une grande communauté qui facilite sa prise en main.

Les Dockerfiles servent à décrire comment construire les conteneurs. Nous avons choisi d'en créer quatre : deux pour la phase de développement back et front et deux pour la partie production. Les Dockerfiles de développement incluent des outils qui permettent, par exemple de voir immédiatement les changements dans le code. Ceux de production, en revanche, sont plus léger et optimisés pour être rapide et sécurisé. Cette séparation permet d'avoir des environnements adaptés à chaque besoin tout en gardant une base commune.

Les Dockerfiles facilitent notre travail de développeurs. Pour le backend, ils permettent de configurer rapidement un environnement avec Deno (framework de développement) et toutes ses dépendances. Pour le frontend, ils assurent que tout le monde travaille avec les mêmes versions de React ou Next.js, que ce soit en développement ou en production. Avec les Dockerfiles, il n'y a pas de surprise !

Les alternatives :

Podman

Les +
Pas besoin d'accès root
Compatible avec les commandes Docker

Les -
Moins d'outils disponibles
Communauté et écosystème moins développés

Vagrant

Les +
Permet de créer des environnements isolés basés sur des machines virtuelles
Parfait pour simuler des environnements proches de la production

Les -
Gourmand en ressources
Temps de démarrage plus long
Pas adapté aux workflows CI/CD modernes

Outil : GitLab

Documentation : https://docs.gitlab.com/

GitLab est une solution robuste qui combine la gestion de code et des pipelines CI/CD dans un même environnement. Cette intégration native réduit la complexité et garantit une fluidité dans l'exécution des workflows.

Les GitLab Runners jouent un rôle clé pour exécuter les jobs CI/CD. Ils offrent une flexibilité essentielle pour adapter les pipelines à nos besoins spécifiques. GitLab propose une personnalisation poussée, ce qui est un atout dans un projet nécessitant des workflows sur mesure.

Pour ce projet, Gitlab répond parfaitement à nos objectifs en centralisant efficacement le code, les pipelines et la collaboration entre les membres de l'équipe.

Stratégie de déploiement : Blue-Green

Documentation: https://www.lemagit.fr/definition/Deploiement-bleu-vert / https://www.free-work.com/fr/tech-it/blog/metiers-it/blue-green-devops-une-approche-efficace-pour-des-deploiements-sans-faille / https://judoscale.com/blog/heroku-preboot

Le déploiement Blue-Green est une méthode qui permet d'introduire une nouvelle version d'une application sans perturber les utilisateurs. Elle repose sur deux environnements identiques :
L'environnement "Blue", qui correspond à la version actuelle en production
L'environnement "Green", où est déployée la nouvelle version
Une fois que la version Green est prête et validée, le trafic utilisateur est basculé de Blue à Green grâce à un load balancer. Si un problème est détecté, le basculement peut être annulé rapidement, permettant de revenir à l'état précédent sans impact sur les utilisateurs.

Cette stratégie est particulièrement avantageuse pour le projet. Elle garantit une continuité de service, élément essentiel pour offrir une expérience utilisateur sans interruption. Cette méthode s'intègre parfaitement à notre pipeline CI/CD. Après avoir déployé la nouvelle version dans l'environnement Green, passé les tests automatisés,

une étape de validation manuelle peut être ajoutée avant de basculer le trafic. Cela offre un contrôle supplémentaire et renforce la fiabilité du déploiement. En combinant cette stratégie avec des outils comme Docker et GitLab, nous disposons d'un processus de déploiement à la fois sûr, rapide et adaptable.

Les alternatives :

Canary Deployment

Les +

Permet de déployer la nouvelle version à un petit pourcentage d'utilisateurs avant un déploiement complet
Idéal pour détecter les bugs avec un faible impact

Les -

Plus complexe à mettre en place (besoin d'un contrôle fin du trafic).
Nécessite un suivi précis des métriques en temps réel
Blue-Green est plus simple à configurer pour ce projet, tout en garantissant un retour rapide à l'état précédent en cas de problème
Il faut également une population d'utilisateur/machine élevé pour voir si tout ce passe correctement ou non

Rolling Deployment

Les +

Déploie progressivement la nouvelle version en remplaçant les instances une par une
Évite de dupliquer les environnements (moins de ressources nécessaires)

Les -

Plus risqué si une mise à jour défectueuse est propagée
Pas de rollback instantané, car les anciennes versions sont remplacées progressivement
Le Rolling Deployment n'offre pas la même sécurité qu'un basculement instantané comme Blue-Green, surtout pour des applications où la continuité est cruciale.

Hébergement : DigitalOcean

Documentation : https://www.digitalocean.com/docs

DigitalOcean est une plateforme cloud mature et largement adoptée, reconnue pour sa simplicité et sa flexibilité. Elle fournit des outils puissants pour la gestion des environnements et l'ajustement des ressources, adaptés aux besoins spécifiques de notre projet.

En choisissant DigitalOcean, nous avons opté pour une solution plus performante et économique, avec une maîtrise accrue des coûts et une optimisation de la performance. Ce choix nous permet de bénéficier d'une infrastructure moderne, évolutive, et adaptée aux besoins techniques et financiers de notre projet.

Outil de test Front : Cypress

Documentation: https://docs.cypress.io/app/end-to-end-testing/writing-your-first-end-to-end-test

Cypress est l'outil sélectionné pour automatiser les tests frontend dans ce projet. Il permet de vérifier le bon fonctionnement de l'interface utilisateur, en simulant les interactions des utilisateurs comme les clics, les saisies ou la navigation. Cypress est reconnu pour sa rapidité, sa simplicité d'utilisation, et sa capacité à fournir des retours en temps réel lors de l'exécution des tests.

Cypress offre des fonctionnalités clés comme les tests end-to-end, les tests d'intégration et les tests unitaires des composants frontend. Intégré à la pipeline CI/CD, il garantit une détection rapide des anomalies et évite de déployer des versions défectueuses en production. En utilisant Cypress avec notre stack React et Next.js, nous assurons une qualité élevée et une expérience utilisateur fluide.

Outil de Supervision et monitoring : Grafana

Documentation: https://grafana.com/docs/

Grafana permet de créer des tableaux de bord qui facilitent le suivi des applications et de l'infrastructure en temps réel. C'est un outil fiable et bien documenté.

Grafana surveille les performances des services backend et frontend (temps de réponse, erreurs HTTP), les logs des conteneurs Docker, et l'état des GitLab Runners. Nous envisageons également un tableau de bord adapté aux métriques clés des pipelines CI/CD et des environnements Blue-Green, avec des alertes pour les seuils critiques.

Son intégration avec Docker et les GitLab Runners étend la supervision au-delà des applications, permettant de suivre l'état des conteneurs et les performances des jobs CI/CD. Cela garantit une visibilité complète sur l'ensemble du pipeline, de la construction des images à leur déploiement.

Développement backend

Technologie principale : Deno

Documentation: https://docs.deno.com/

Deno a été choisi pour ce projet en raison de ses caractéristiques modernes, comme l'intégration native avec TypeScript et un modèle de sécurité renforcé. Contrairement à Node.js, Deno impose des permissions explicites pour accéder aux ressources système, réduisant ainsi les risques de vulnérabilités.

Deno est particulièrement pertinent pour ce projet grâce à sa légèreté et ses performances élevées, idéales pour des API backend nécessitant des temps de réponse rapides. Sa conception moderne s'intègre facilement dans un pipeline CI/CD, offrant une expérience fiable et adaptée.

Les alternatives :

Node.js

Les +
Écosystème mature avec un grand nombre de bibliothèques disponibles via npm
Large communauté et support actif
Compatible avec presque tous les services et outils DevOps

Les -
Absence de sécurité native (permissions système ouvertes par défaut)
Gestion des dépendances plus complexe (problèmes de versions multiples)
Pas de support natif pour TypeScript, nécessitant des outils supplémentaires comme ts-node ou des configurations spécifiques

Python (Flask)

Les +
Simple à apprendre et à utiliser, idéal pour les projets rapides ou les MVP
Large communauté et nombreuses ressources pour résoudre les problèmes

Les -
Moins performant pour les applications nécessitant une gestion intensive des entrées/sorties (I/O).
Non conçu spécifiquement pour les workflows modernes de type DevOps.

Serveur : Hono

Documentation: https://hono.dev/docs/getting-started/basic

Hono est un framework léger et performant qui simplifie la création d'API et d'applications web. Il offre des fonctionnalités modernes comme la gestion des middlewares, la compatibilité avec TypeScript, et une structure modulaire adaptée aux projets évolutifs.

Hono s'intègre parfaitement avec Deno, ce qui garantit des performances élevées et une maintenance simplifiée. Sa rapidité et sa légèreté en font un excellent choix pour des projets où les temps de réponse sont critiques.

Développement frontend

Technologie principale : React avec Next.js

Documentation: https://legacy.reactjs.org/docs/getting-started.html / https://nextjs.org/docs

React et Next.js ont été choisis pour ce projet en raison de leur combinaison unique de flexibilité et de performance. React, grâce à sa structure basée sur les composants, permet de créer des interfaces utilisateur dynamiques et réactives. Next.js, en complément, apporte des fonctionnalités avancées comme le rendu côté serveur (SSR) et la génération statique (SSG), garantissant une meilleure performance et un référencement optimisé.

React et Next.js répondent parfaitement aux besoins de ce projet en offrant une structure modulaire et une grande extensibilité. Le rendu côté serveur de Next.js améliore considérablement les temps de chargement des pages, un critère essentiel pour garantir une bonne expérience utilisateur. La compatibilité de Next.js avec

TypeScript s’aligne également avec les technologies backend (Deno), favorisant une cohérence sur l’ensemble de la stack.

React et Next.js s’imposent comme un choix équilibré, offrant à la fois la flexibilité de React et les optimisations puissantes de Next.js, parfaitement adaptées aux objectifs techniques et de performance du projet.

Les alternatives :

Vue.js :

Les +
Plus simple à prendre en main pour les débutants.

Les -
Moins performant que Next.js pour le rendu côté serveur.

Angular :

Les +
Framework complet avec une approche "tout-en-un".

Les -
Courbe d’apprentissage plus élevée et plus rigide dans sa structure.

Bibliothèque UI : MUI

Documentation: https://mui.com/

Material-UI (MUI) a été choisi pour ce projet en raison de sa flexibilité et de sa compatibilité avec React. Inspirée des principes de design Material Design de Google, MUI offre un ensemble complet de composants prêts à l’emploi, permettant de créer rapidement des interfaces modernes et cohérentes. Son intégration fluide avec TypeScript renforce également sa pertinence pour un projet axé sur la qualité du code et la maintenabilité.

Les alternatives :

Bootstrap :

Les +
Large adoption et support étendu pour le responsive design

Les -
Styles par défaut plus rigides, nécessitant souvent des surcharges CSS

Tailwind CSS :

Les +
Approche utilitaire très flexible, idéale pour créer des designs personnalisés.

Les -
Nécessite un temps d'apprentissage initial plus long et produit des classes CSS parfois complexes à maintenir.

Développement mobile

Technologie principale : React Native

Documentation : https://reactnative.dev/ et https://www.npmjs.com/package/react-native-vision-camera

React Native est un framework populaire qui permet de développer des applications mobiles pour Android et iOS à partir d'une seule base de code. Il se distingue par un développement rapide grâce à sa base de code unique et au hot-reloading, les développeurs peuvent apporter des modifications instantanées et tester rapidement les changements.

Comme l'équipe utilise déjà React pour le frontend web, elle peut facilement réutiliser ses compétences, réduisant ainsi la courbe d'apprentissage.

React Native offre une intégration fluide avec l'appareil photo grâce à des bibliothèques comme react-native-camera ou react-native-vision-camera. Ces outils permettent d'accéder aux fonctionnalités natives de l'appareil photo, comme le scan de codes-barres.

Les alternatives :

Flutter :

Les +
Meilleure performance grâce à son moteur graphique propriétaire.
Interface utilisateur extrêmement personnalisable.

Les -
Base de code en Dart, nécessitant un apprentissage pour de nombreux développeurs.
Applications souvent plus volumineuses en taille.

Kotlin/Swift natifs :

Les +
Performances optimales, adaptées aux projets exigeants en termes de complexité ou d'accès matériel (ex. : jeux, AR/VR).
Support direct des plateformes Android et iOS.

Les -
Développement distinct pour chaque plateforme, augmentant le temps et les coûts.
Moins adapté pour les projets avec des cycles rapides.

Outil de développement : Metro

Documentation : https://reactnative.dev/docs/metro

Metro est le bundler par défaut de React Native, utilisé pour transformer et optimiser le code JavaScript en un format exécutable pour les applications mobiles. C'est un choix naturel pour les projets utilisant React Native en raison de son intégration native et de ses performances optimales.

Conclusion

Les choix technologiques du projet sont en parfaite adéquation avec ses objectifs. En adoptant des outils modernes comme React Native, Deno, et Docker, nous assurons une prise en main rapide pour les équipes tout en garantissant une compatibilité avec les standards professionnels.

Ces solutions permettent de répondre aux besoins actuels tout en offrant une grande évolutivité, essentielle pour accompagner la croissance du projet. Elles allient simplicité, performance, et scalabilité, garantissant une plateforme performante et durable.