***Compte rendu des modifications***

*Contexte :*

Ce projet est un proof of concept (POC) pour la mise en place d'une application multi-plateforme (web et desktop dans mon cas, mais peut être mobile, embarqué, etc…).

Le but de mon application est de permettre à un utilisateur de gérer une cartographie de port commercial.

L'utilisateur pourra alors modifier les emplacements des conteneurs, les déplacer, les supprimer, les ajouter, etc...

**Tests d'Intégration et Unitaires de l'Application - Méthodologie et Exemple**

Description :

Au cours du semestre 6 de la SAÉ, j'ai réalisé une série de tests d'intégration et unitaires pour évaluer la robustesse, la fiabilité et la qualité de l'application développée. Ces tests ont été conçus pour vérifier le bon fonctionnement des différentes fonctionnalités de l'application, en mettant l'accent sur la cohérence de l'interface utilisateur, la précision des fonctionnalités métier et la stabilité du code.

Méthodologie :

La réalisation des tests d'intégration et unitaires a suivi une approche méthodique, basée sur les bonnes pratiques de test logiciel. Voici les étapes clés de la méthodologie employée :

1. Analyse des exigences : Avant de commencer les tests, j'ai effectué une analyse approfondie des exigences fonctionnelles et techniques de l'application, en identifiant les cas d'utilisation principaux et les scénarios de test associés.

2. Conception des tests : En fonction des exigences identifiées, j'ai conçu une suite de tests couvrant les différentes parties de l'application, en se concentrant à la fois sur les tests unitaires pour les composants individuels et sur les tests d'intégration pour les interactions entre les composants.

3. Implémentation des tests : Les tests ont été implémentés en utilisant des frameworks de test appropriés, tels que Jest et @testing-library/react pour les tests d'intégration de l'interface utilisateur, et Mocha et Chai pour les tests unitaires des services et des API.

4. Exécution des tests : Les tests ont été exécutés de manière régulière tout au long du processus de développement, permettant de détecter les erreurs et les anomalies dès leur apparition et de les corriger rapidement.

5. Analyse des résultats : Les résultats des tests ont été analysés de manière critique pour identifier les zones à risque et les problèmes potentiels, et pour guider les efforts de correction et d'amélioration du code.

Exemple de Test Unitaire :

Une image contenant texte, capture d’écran

Description générée automatiquementVoici un exemple de test unitaire que j'ai développé pour évaluer le bon fonctionnement des opérations CRUD sur les entités « rôles » de l’API REST :

Exemple de Test d'Intégration :

Une image contenant texte, capture d’écran

Description générée automatiquementVoici un exemple de test d'intégration que j'ai développé pour évaluer le bon rendu d'une page de création de zone dans l’application React :

La réalisation des tests d'intégration et unitaires a été un élément essentiel du processus de développement de l'application, contribuant à assurer sa qualité, sa fiabilité et sa stabilité. En suivant une approche méthodique et en utilisant des outils appropriés, j'ai pu valider efficacement les différentes fonctionnalités de l'application, garantissant ainsi une expérience utilisateur optimale.

**Réalisation du CI/CD**

J'ai également mis en place un processus de CI/CD (Continuous Integration/Continuous Deployment) pour assurer un déploiement efficace et automatisé de l'application développée. Cette démarche a été cruciale pour garantir une gestion efficace du cycle de vie du logiciel, en automatisant les phases de construction, de test et de déploiement de l'application.

La réalisation du CI/CD s'est déroulée en plusieurs étapes, comme décrit ci-dessous :

1. Création des Dockerfiles : Dans un premier temps, j'ai créé des Dockerfiles pour chaque composant de l'application, notamment l'API, l'application web et la documentation associée. Cette étape a permis de containeriser les différents services de l'application, facilitant ainsi leur déploiement et leur gestion.

2. Docker Compose : Un fichier Docker Compose à été fait pour orchestrer les différents conteneurs de l’application. Ce fichier a permis de définir les services nécessaires et leurs dépendances, facilitant ainsi le déploiement et la gestion de l’ensemble de l’application.

3. Mise en place du fichier .gitlab-ci.yml : Ensuite, j'ai configuré un fichier .gitlab-ci.yml pour mettre en place le processus de CI/CD sur l'environnement de développement utilisé, à savoir Etulab. Ce fichier a défini les différentes étapes du pipeline CI/CD, telles que la construction, les tests et le déploiement de l'application.

4. Intégration des tests dans le CI/CD : J'ai intégré les tests unitaires et d'intégration dans le pipeline CI/CD afin de garantir la qualité du code à chaque étape du processus de développement. Ces tests ont été exécutés automatiquement à chaque modification du code source, permettant ainsi de détecter rapidement les éventuelles erreurs ou anomalies.

5. Configuration du serveur pour le CI/CD : En parallèle, j'ai configuré les dossiers, fichiers et paramètres nécessaires sur le serveur de déploiement pour permettre l'intégration continue et le déploiement automatique de l'application. Cette configuration a assuré une gestion efficace du processus de CI/CD dans un environnement de production.

6. Déploiement automatique avec CI/CD : Enfin, j'ai mis en place la copie des fichiers vers le serveur de déploiement à travers le pipeline CI/CD. Cela a permis de lancer automatiquement le docker-compose et de déployer l'application dès la fin du processus de construction et de test.

La mise en place du CI/CD a été un élément clé du processus de développement de l'application, permettant d'automatiser les tâches répétitives, d'améliorer la qualité du code et d'accélérer le cycle de vie du logiciel. En adoptant cette approche, j'ai pu garantir une livraison continue et fiable de l'application, répondant ainsi aux exigences de qualité et de performance attendues.

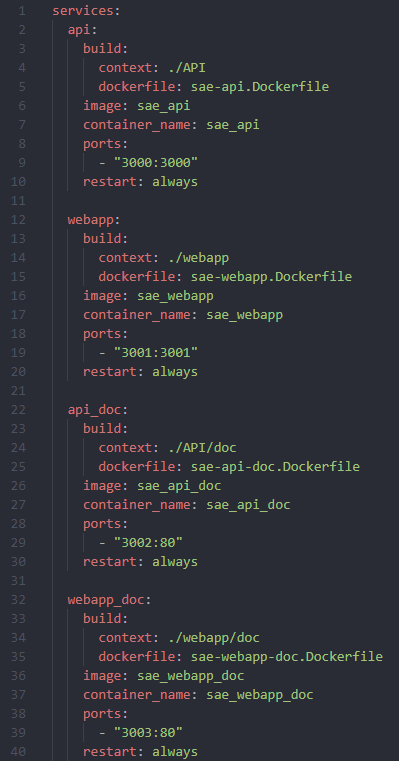
**Exemples de réalisation du CI/CD**

1. Création des Dockerfile pour l’API

Une image contenant texte, Police, capture d’écran, conception

Description générée automatiquementVoici le code d’un Dockerfile qui permet de dockeriser l’API.

1. Création d’un fichier Docker Compose

Voici la création du Docker Compose qui permet de réunir et de lancer chaque service correctement.

1. Mise en place du fichier .gitlab-ci.yml pour le pipeline CI/CD

Une image contenant texte, capture d’écran, menu

Description générée automatiquementVoici un extrait (pour l’application web et le déploiement) du fichier permettant de faire chaque étape de la construction de l’application et le déploiement sur le serveur à la fin.