ALGOFLEX

Guide de développement

Introduction

Dans ce document nous allons répertorier les bonnes pratiques de développement dans l'objectif d'uniformiser le code de notre application Algoflex. Dans un souci de maintenabilité chaque développeur sera tenu de prendre connaissance des règles & informations ci-dessous.

Développement Front End

Langage & bonnes pratiques

Nous utilisons le framework « React » couplé avec le langage « Typescript » pour le développement du front-end. Pour les tests unitaires nous avons utilisé « testing-library »

Les composants sont rassemblés dans le dossier « components ». Chaque implémentation des composants sont stockées avec le fichier de test correspondant dans un dossier ayant le nom du composant. Les composants sont des éléments réutilisables dans l'application.

Les vues sont rassemblées dans le dossier « views » de la même façon que les composants. Les vues sont les composants finaux qui seront appelé lorsqu'on souhaite accéder à l'une des pages du site.

Tests unitaires

Chaque composant react doit être testé de plusieurs manières :

- Test de render avec création de snapshot
- Test d'affichage des données
- Test de redirection si applicable

Mockups

Des Mockups sont disponible pour aider au développement de la partie graphique de l'application AlgoFlex. Ils vous seront communiqués en fonction de vos tâches si applicable. Ces mockups servent de base de travail et peuvent être modifié / amélioré après avoir été validé par l'équipe porteur de projet.

Développement Back End

Langage & bonnes pratiques

Nous utilisons le framework « NestJS » couplé avec le langage « Typescript » pour le développement du back-end. Les tests unitaires sont réalisés grâce à la libraire « Jest ». Nous utilisons « TypeORM » pour faciliter la création et l'accès à la base de données.

Tests unitaires

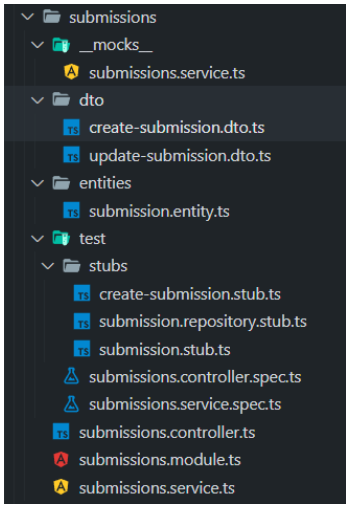
Les tests unitaires ont été réalisés avec le framework « Jest », celui-ci permet de facilement créer des « mocks » des classes et fonctions. Le mock est le processus utilisé dans les tests unitaires lorsque l'unité testée a des dépendances externes, l'objectif étant de simuler le comportement des dépendances pour isoler l'élément à tester.

La stratégie utilisée pour nos tests est de créer des mocks pour les éléments du « MVC », le modèle, vue, contrôleur.

Pour un module qui contient un contrôleur, un service et une entité qui lui est associée, les étapes pour créer nos tests sont :

1. Créer des mock du DTO, l'objet de transfert de données, pour les actions de création et de modification, ces mocks contiendront les données de test que l'on utilise pour la suite.
2. Créer un mock du « Repository », c'est une classe qui permet d'avoir une couche d'abstraction entre la logique métier et la partie accès de données.
3. Créer ensuite un mock du « Service » qui s'occupe de l'aspect logique métier.
4. Pour finir créer les fichiers de tests « spec » pour le contrôleur et le service, ces tests utiliseront les mocks crée plutôt.
5. Chaque fonction testée devrait avoir une fonction de test qui lui est associée. Une dernière fonction de test doit faite pour tester les exceptions.

L'architecture finale pour les tests unitaires doit ressembler à l'image ci-dessous :



Actuellement la couverture de code est de 79%.

All files

79.08% Statements 363/459 **81.81%** Branches 36/44 **81.57%** Functions 62/76 **77.83%** Lines 323/415

Déploiement & mise en production

L'application Algoflex comporte plusieurs modules internes et externes :

Modules internes :

- Serveur Front End
- Serveur Back End et d'autocomplétion
- Docker API Engine
- Base de données

Modules Externes :

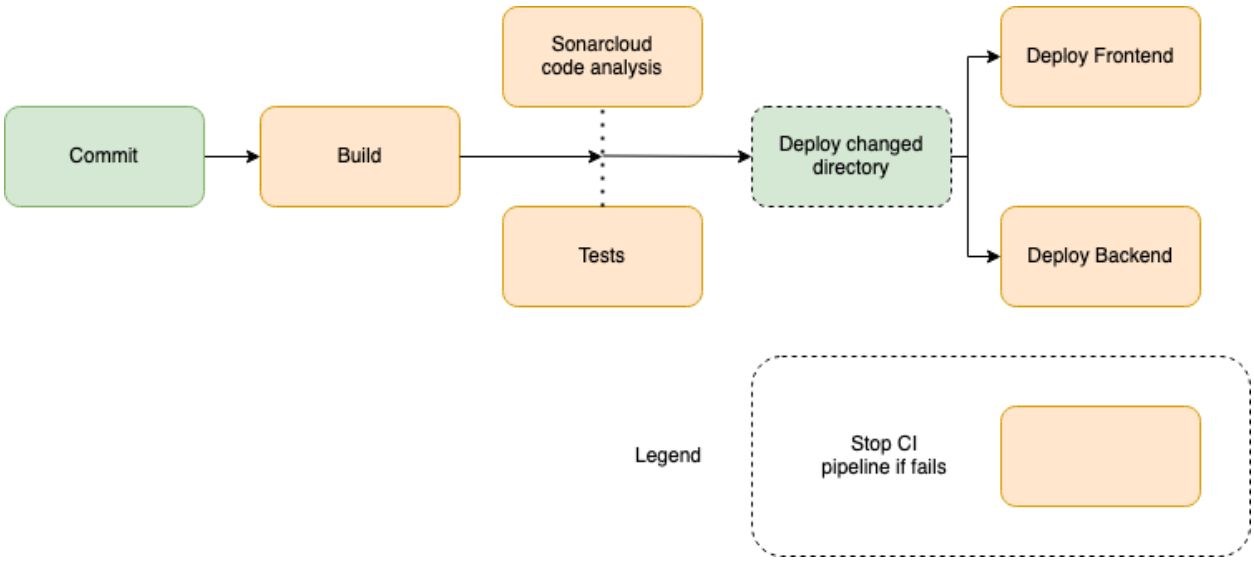
- Hébergement Netlify (Front End)
- Hébergement Azure (Docker API Engine)
- Hébergement Heroku (Back End & Base de données)
- CI / CD GitHub (déploiement et intégration continues)
- SonarCube (Revue de code)

CI/CD

Le CI/CD utilise les outils mis à disposition de GitHub pour la compilation de l'application et le passage des tests. Nous y avons ajouté une revue de code effectuée par un outil externe Sonarcube qui nous permet de vérifier la qualité du code et de mettre en warning des éléments critiques liés aux codes smells, security hotspot, bug etc.

La validation du processus d'intégration continue est soumise à la conformité du code aux règles définies. Une « Merge request » ne peut être effectuée que si tous les éléments de compilation et de tests sont valides.

Ci-dessous le schéma de validation du CI/CD comme établi dans GitHub.

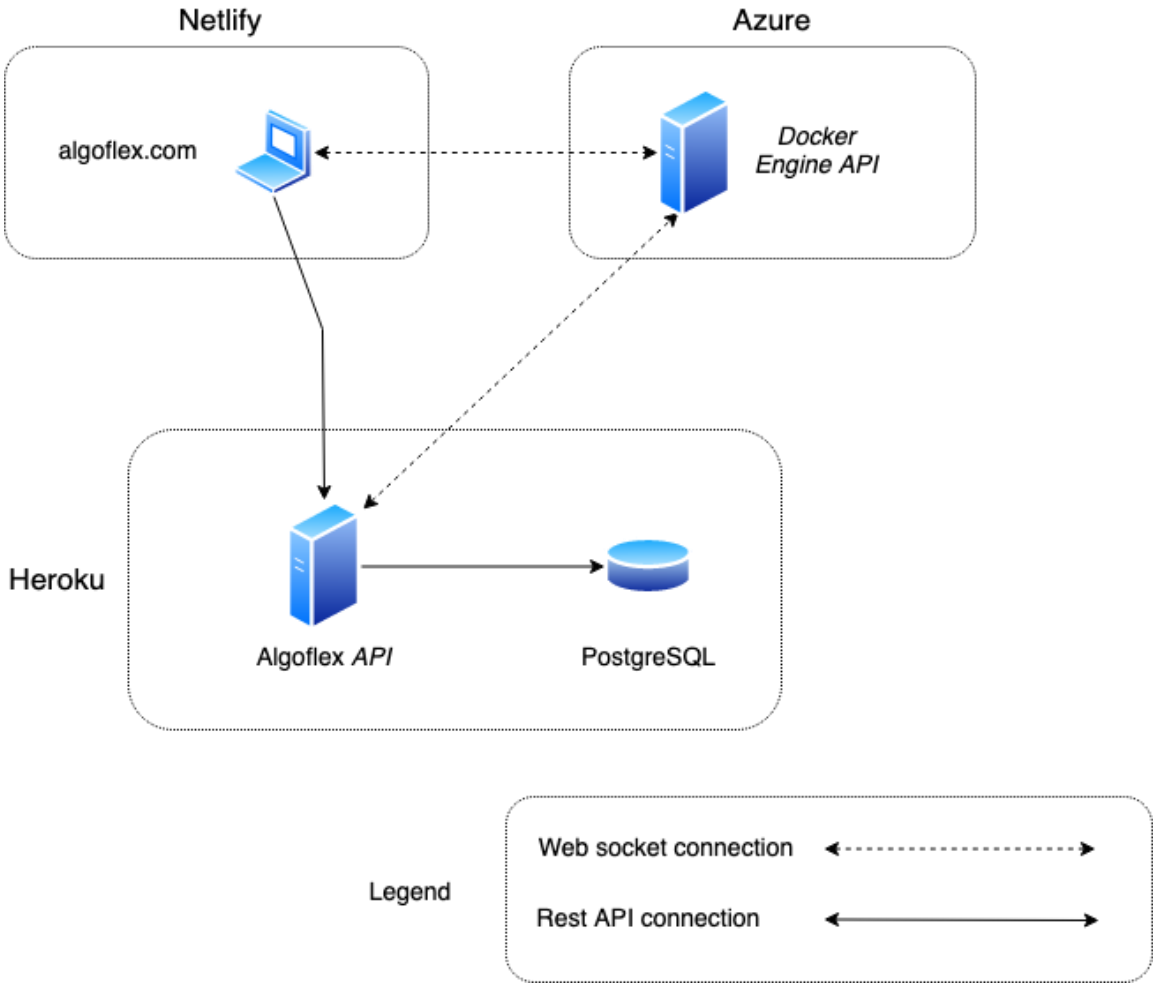


Infrastructure

Notre infrastructure est divisée sur trois services d'hébergements différents :

- Le front End hébergé sur Netlify
- Le back End hébergé sur Heroku
- Le Docker API Engine hébergé sur Azure

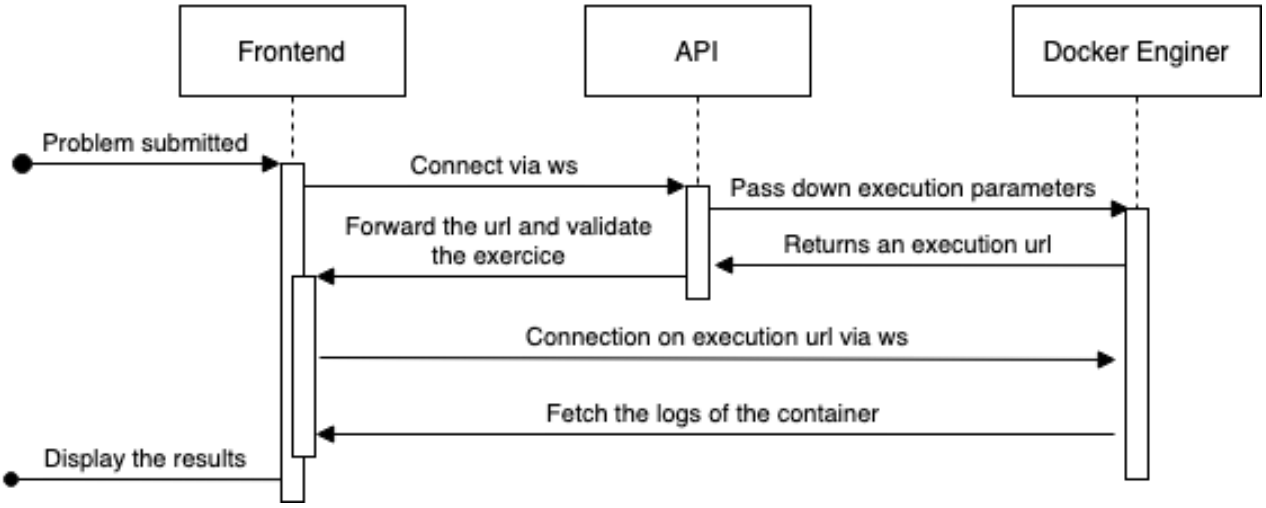
Chaque service communique soit via des requêtes d'API REST soit via WebSocket.



Exemple fonctionnel

Ci-dessous le schéma fonctionnel de la soumission d'un problème par l'utilisateur et l'interaction des différents modules d'Algoflex.

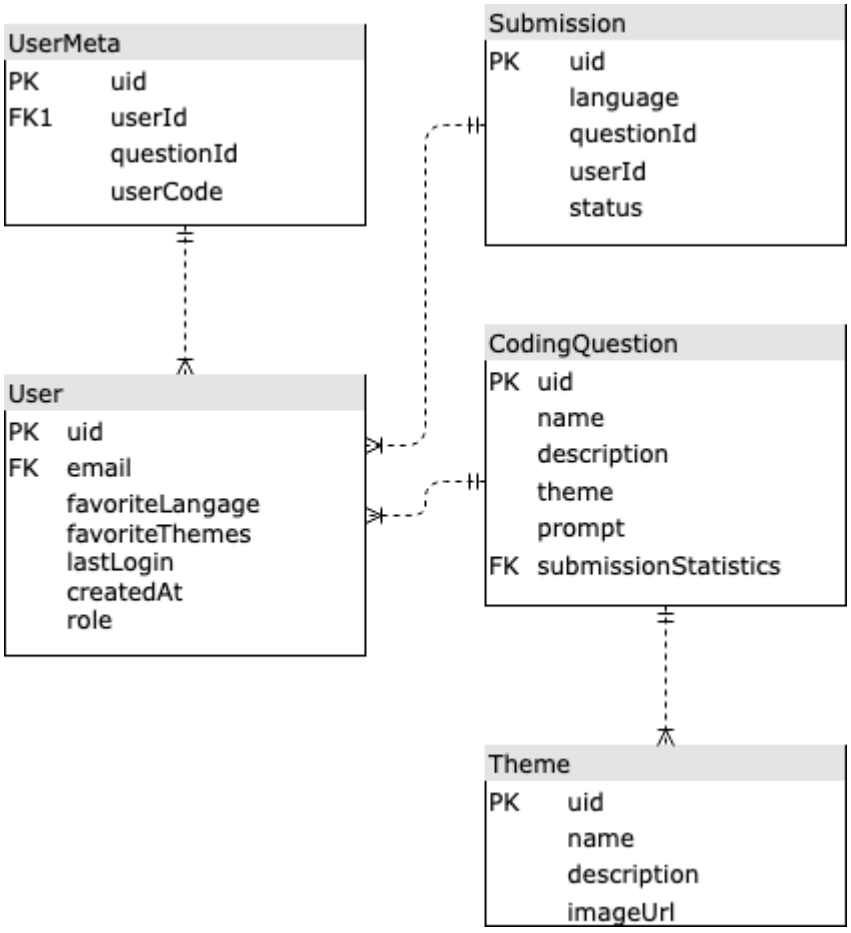
*Nb: **ws** stands for **WebSocket***



Mise en place Base de données

Base de données principale

Nous utilisons PostgreSQL pour notre base de données SQL. Cette base de données contient toutes les informations nécessaires au fonctionnement de l'application (thèmes, exercices, soumission utilisateur etc).



Base de données & service d'authentification

Le stockage des utilisateurs et les services d'authentification passent via le module firebase de Google qui gère l'inscription, le login et la récupération de mot de passe. Ce module externe nous permet de faciliter le processus et en même temps de sécuriser le stockage des données.