**Rapport Projet : Fullstack User Management**

# Présentation :

Ce projet est une application web fullstack permettant la gestion des utilisateurs (ajout, modification, suppression et affichage). Il a été développé dans le cadre d’un projet pédagogique visant à mettre en pratique les concepts de développement web, d’intégration continue (CI/CD), de conteneurisation avec Docker, et de tests automatisés.

L’objectif principal est de créer une solution complète, maintenable et déployable, en combinant les technologies modernes telles que Node.js, Express, MySQL, React.js et Docker. Le projet est conçu pour être facilement déployé en production avec une infrastructure évolutive et automatisée.

[Présentation : 1](#_Toc195223500)

[Partie 1 : Mise en place du projet 3](#_Toc195223501)

[**1.** **Backend (Express.js, MySQL, Docker)** 3](#_Toc195223502)

[**2. Frontend (React.js, Docker)** 4](#_Toc195223503)

[**3.Explication de la Base de Données** 5](#_Toc195223504)

[Partie 2 : Dockerisation et Configuration de Docker Compose 6](#_Toc195223505)

[**1. Dockerisation du Backend** 6](#_Toc195223506)

[**2. Dockerisation du Frontend** 6](#_Toc195223507)

[**3. Configuration de Docker Compose** 6](#_Toc195223508)

[**4. Lancer l’ensemble de l’application** 6](#_Toc195223509)

[Partie 3: Tests 7](#_Toc195223510)

[**1. Tests Unitaire** 8](#_Toc195223511)

[**2. Tests d'Intégration** 8](#_Toc195223512)

[**3. Couverture de Code** 8](#_Toc195223513)

[**4. Exécution des Tests Locaux** 8](#_Toc195223514)

[**5. Rapport de Couverture HTML** 8](#_Toc195223515)

[**6. Résultat des Tests** 9](#_Toc195223516)

[Partie 4 : GitHub Actions – CI/CD 9](#_Toc195223517)

[**1. Création du fichier .github/workflows/ci.yml** 9](#_Toc195223518)

[**2. Contenu du fichier ci.yml** 10](#_Toc195223519)

[**3. Configuration des secrets GitHub** 10](#_Toc195223520)

[**4. Déclenchement du workflow** 10](#_Toc195223521)

[**5. Résultat attendu** 10](#_Toc195223522)

[Difficultés rencontrées et Solutions apportées 11](#_Toc195223523)

[Conclusion 11](#_Toc195223524)

# Partie 1 : Mise en place du projet

Le projet **Fullstack User Management** a été développé pour gérer les utilisateurs via une application web. Le système est divisé en deux parties principales : le backend et le frontend.

## **Backend (Express.js, MySQL, Docker)**

**Technologies utilisées** :

* Express.js pour le backend : Un framework minimaliste et flexible pour Node.js, permettant de créer des API RESTful rapidement.
* MySQL pour la gestion de la base de données : Un système de gestion de base de données relationnelle pour stocker les informations des utilisateurs.
* Docker pour la conteneurisation : Docker permet de créer des environnements d'exécution isolés et portables.

**Fonctionnalités du backend :**

* Création, récupération, mise à jour et suppression des utilisateurs via des API RESTful.
* Connexion sécurisée à la base de données MySQL.
* Gestion des erreurs et des réponses API.
* Routes protégées pour certaines actions (par exemple, l’ajout d’un utilisateur).

**Fichier principal :** server.js

**Exemple d’API :**

* **GET /users :** Récupérer la liste des utilisateurs.
* **POST /users :** Ajouter un nouvel utilisateur.
* **PUT /users/:id :** Mettre à jour un utilisateur existant.
* **DELETE /users/:id :** Supprimer un utilisateur.

**Commandes :**

* Pour l’ Initialisation du projet : **« npm init -y »**
* Pour l’ Installation des dépendances : **« npm install express mysql2 dotenv cors »**

## **2. Frontend (React.js, Docker)**

**Technologies utilisées :**

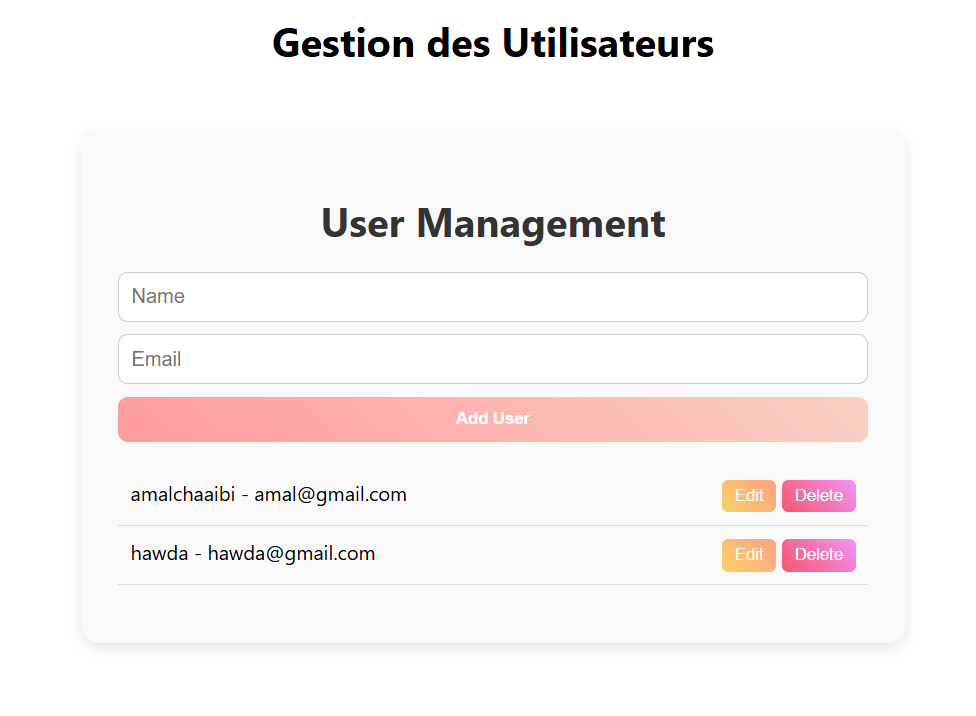
* + **React.js** pour le développement du frontend : Une bibliothèque JavaScript pour construire des interfaces utilisateur dynamiques.
  + **Docker** pour la conteneurisation du frontend afin d’assurer une gestion cohérente des environnements de développement et de production.
* **Fonctionnalités du frontend :**
  + Formulaire permettant d’ajouter, modifier ou supprimer des utilisateurs.
  + Affichage de la liste des utilisateurs récupérée depuis l’API backend.

**Exemple d'interface utilisateur :**

* + Page de connexion pour les utilisateurs.
  + Formulaire pour ajouter un utilisateur.
  + Liste des utilisateurs avec options pour modifier et supprimer.

**Commandes :**

* Création avec Vite : **« npm create vite frontend --template react »**
* Installation des dépendances : **« npm install axios react-router-dom »**



## **3.Explication de la Base de Données**

* SGBD : MySQL
* La table users permet de stocker les informations des utilisateurs. Elle est constituée de quatre colonnes principales. La colonne **id** est de type entier (INT) et est auto-incrémentée (AI), ce qui signifie qu'elle s'incrémente automatiquement à chaque nouvel utilisateur ajouté, garantissant ainsi un identifiant unique pour chaque enregistrement. Ensuite, la colonne **name** est de type VARCHAR(100) et permet de stocker le nom de l'utilisateur, avec une longueur maximale de 100 caractères. La colonne **email** est également de type VARCHAR(100) et sert à enregistrer l’adresse email de l’utilisateur. Cette structure permet ainsi d'assurer un enregistrement clair, organisé et précis des informations essentielles des utilisateurs.

# Partie 2 : Dockerisation et Configuration de Docker Compose

Le projet a été dockerisé pour faciliter le déploiement et garantir que l’environnement de développement soit identique à celui de la production. L’utilisation de Docker permet de simplifier l’installation et la gestion des dépendances, tout en créant un environnement stable pour les développeurs.

## **1. Dockerisation du Backend**

Un **Dockerfile** a été créé pour le backend, spécifiant l’image de base (Node.js), l’installation des dépendances, la configuration des variables d’environnement et le démarrage de l’application.

## **2. Dockerisation du Frontend**

Un **Dockerfile** a été créé pour le frontend, spécifiant l’image de base (Node.js), la construction du projet React, et le serveur statique pour servir l’application.

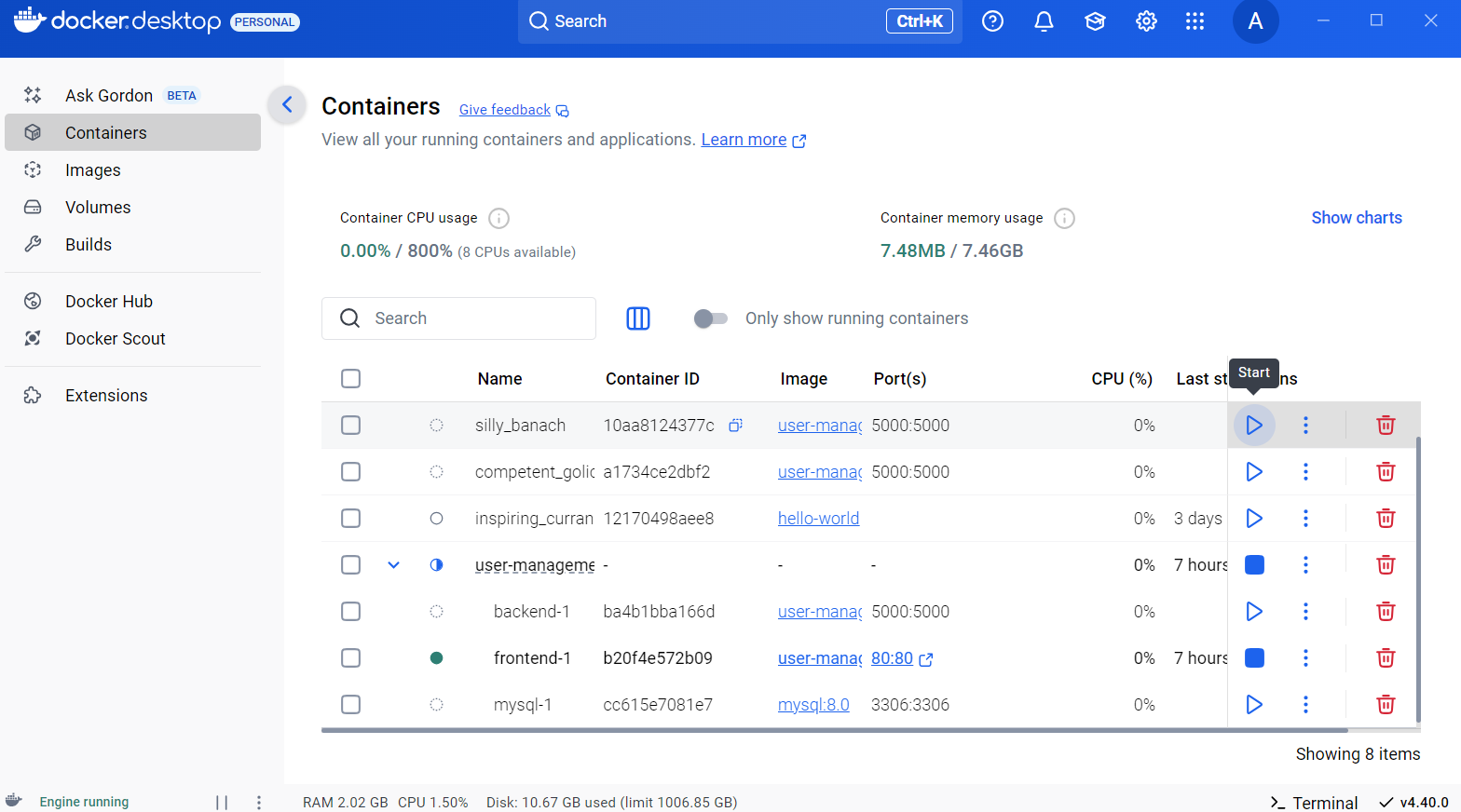
## **3. Configuration de Docker Compose**

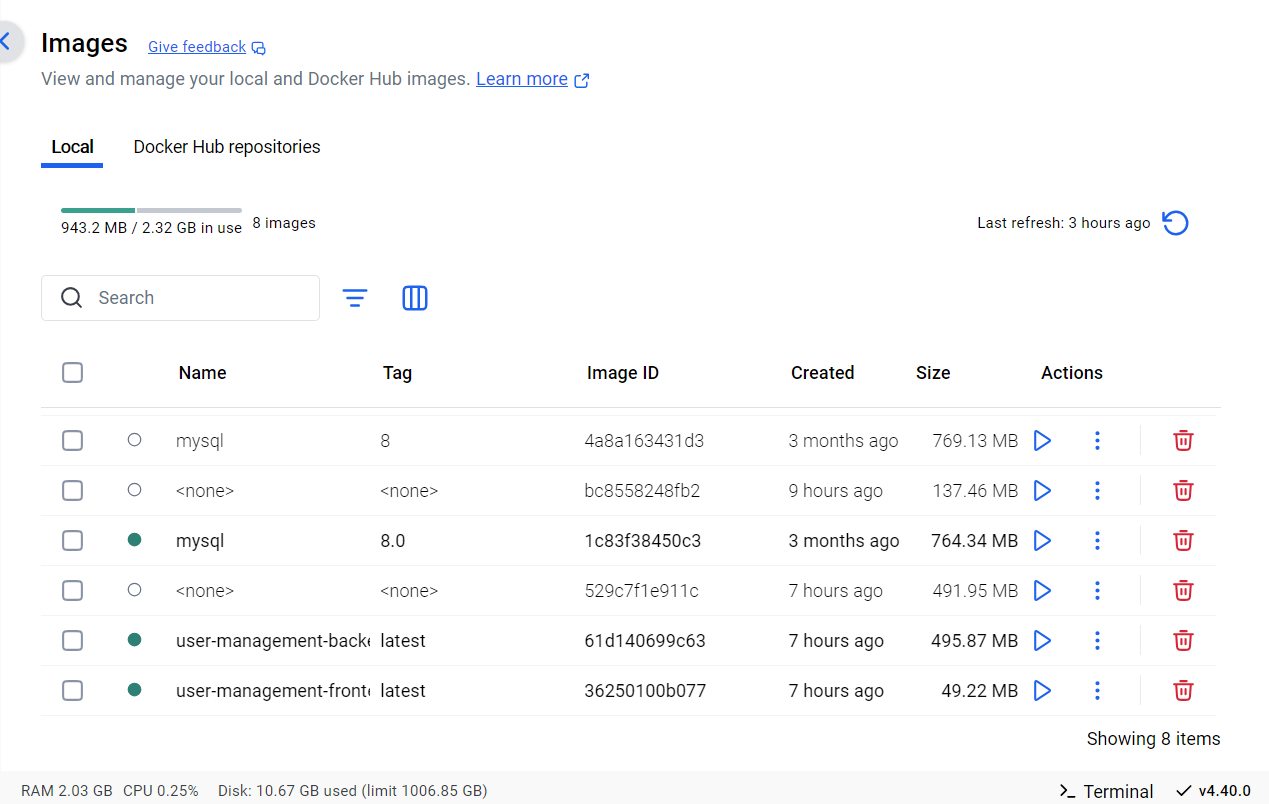
* **Docker Compose** a été utilisé pour orchestrer les conteneurs Docker, en facilitant leur gestion et leur interconnexion.
* Un fichier **docker-compose.yml** a été créé pour définir les services nécessaires (backend, frontend, et base de données MySQL).

## **4. Lancer l’ensemble de l’application**

* Lancer l’application backend, frontend et MySQL avec une seule commande « **docker-compose up ».**

Cette commande reconstruit les images Docker avant de lancer les conteneurs.





# Partie 3: Tests

Dans cette partie, des tests ont été implémentés pour assurer la fiabilité du backend de l'application. Les tests incluent des tests unitaires, des tests d'intégration, et une configuration de couverture de code.

## **1. Tests Unitaire**

Un test unitaire simple a été implémenté pour tester la logique backend de l'application. Un exemple de test unitaire est la vérification d'une fonction qui doit retourner true.

## **2. Tests d'Intégration**

Des tests d'intégration ont été ajoutés pour tester les routes principales de l'application backend. Les tests incluent les requêtes GET, POST, PUT, et DELETE sur la route '/users'. Les tests vérifient si ces routes renvoient les réponses appropriées (statut HTTP 200, 201, etc.).

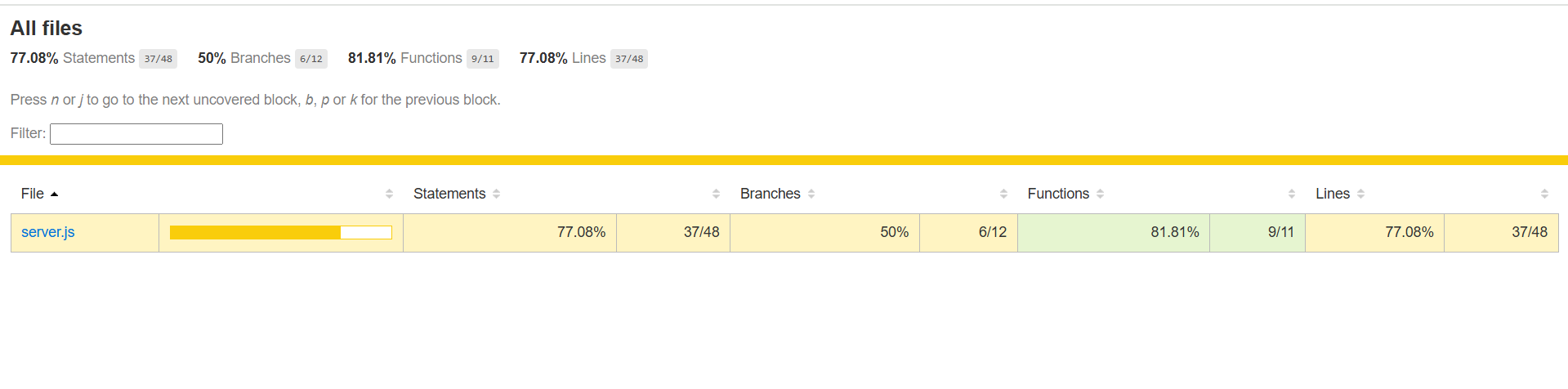
## **3. Couverture de Code**

La couverture de code a été configurée à l'aide de l'outil 'nyc'. Après l'exécution des tests, un rapport de couverture a été généré pour analyser le pourcentage de code testé. Le rapport indique les lignes de code couvertes et non couvertes.

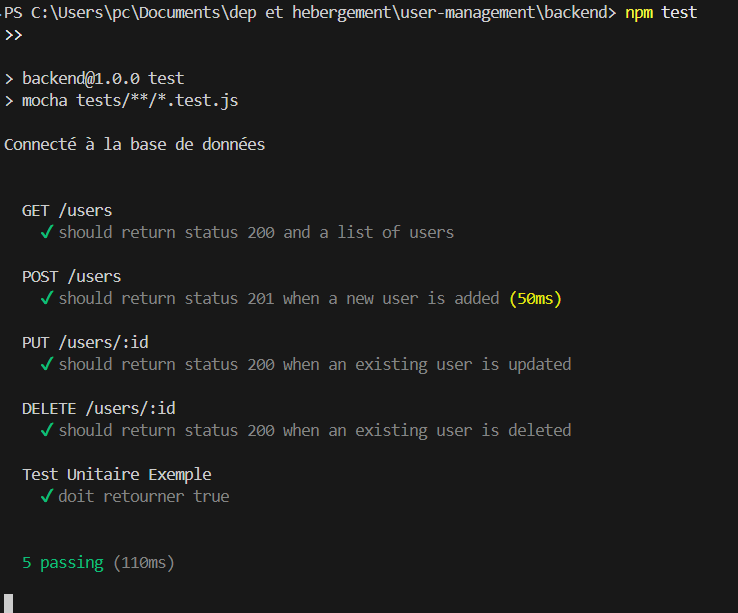
## **4. Exécution des Tests Locaux**

Les tests peuvent être exécutés localement en utilisant la commande suivante dans le terminal : “**npm test”**  
  
Cette commande lance les tests unitaires et d'intégration. Pour obtenir le rapport de couverture, utilisez : **“npm run coverage”**

## **5. Rapport de Couverture HTML**

Après avoir exécuté les tests avec la commande **'npm run coverage'**, un rapport de couverture au format HTML est généré. Vous pouvez visualiser ce rapport dans votre navigateur en ouvrant le fichier 'coverage/index.html'. Ce fichier présente les pourcentages de code couvert par les tests et vous permet d'analyser les lignes non couvertes.

## **6. Résultat des Tests**

Tous les tests ont été exécutés avec succès, comme indiqué ci-dessous :

# Partie 4 : GitHub Actions – CI/CD

Cette partie décrit la mise en place de l'intégration continue et du déploiement continu (CI/CD) via GitHub Actions pour le projet fullstack-user-management.

## **1. Création du fichier .github/workflows/ci.yml**

Un fichier nommé ci.yml a été créé dans le répertoire .github/workflows. Il contient la configuration du pipeline CI/CD.

## **2. Contenu du fichier ci.yml**

Voici les étapes définies dans le pipeline :

1. Récupération du code avec l'action checkout.

2. Setup de Node.js (version 20) pour les tests du backend.

3. Installation des dépendances du backend avec npm install.

4. Lancement des tests avec npm test.

5. Connexion à Docker Hub avec des secrets GitHub.

6. Build et push de l'image Docker du backend vers Docker Hub.

## **3. Configuration des secrets GitHub**

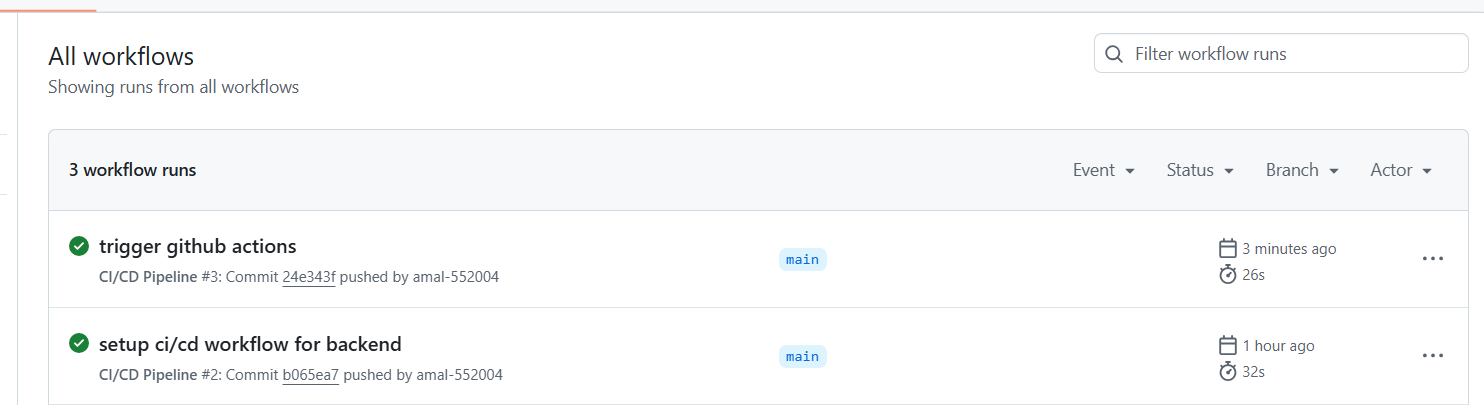
Deux secrets ont été ajoutés dans GitHub :

- DOCKER\_USERNAME : nom d'utilisateur Docker Hub

- DOCKER\_PASSWORD : mot de passe ou token Docker Hub

## **4. Déclenchement du workflow**

Le workflow se déclenche automatiquement lors d’un push ou pull request vers la branche main.



## **5. Résultat attendu**

Une fois le workflow terminé avec succès, l’image Docker du backend est disponible sur Docker Hub dans le dépôt de l’utilisateur.

Cette configuration permet d'automatiser les tests et le déploiement d'images Docker à chaque mise à jour du code source.

# Difficultés rencontrées et Solutions apportées

|  |  |  |  |
| --- | --- | --- | --- |
| | **Problème** |  | | --- | --- | | **Solution** |
| Problème connexion base de données | érification des variables d'environnement et correction de l'URL de connexion |
| Docker ne lance pas MySQL correctement | Ajout de depends\_on dans docker-compose |
| GitHub Actions ne push pas sur Docker Hub | |  | | --- | |  |  |  | | --- | | Création des secrets DOCKER\_USERNAME et DOCKER\_PASSWORD | |

# Conclusion

Pour conclure, ce projet de gestion des utilisateurs a été une expérience très enrichissante qui nous a permis de mettre en pratique plusieurs compétences techniques et outils essentiels dans le développement web moderne. Nous avons pu concevoir et développer un backend robuste avec Express.js, un frontend interactif avec React, et une base de données MySQL permettant de stocker les informations des utilisateurs de manière structurée. La dockerisation du projet nous a permis de mieux comprendre les avantages liés à la conteneurisation des applications, notamment en termes de portabilité et d’environnement unifié. De plus, la mise en place d’un pipeline CI/CD avec GitHub Actions nous a permis d’automatiser plusieurs étapes du cycle de vie du projet : les tests, le build des images Docker et leur publication sur Docker Hub. Malgré certaines difficultés rencontrées, notamment lors de la configuration de Docker, la gestion des variables d'environnement, et la configuration de GitHub Actions, nous avons su trouver des solutions adaptées et progresser. Ce projet constitue donc une base solide que nous pourrions enrichir à l’avenir, en ajoutant par exemple un système d’authentification sécurisé, une interface utilisateur plus complète, ou encore un déploiement automatique sur un serveur distant pour rendre l’application disponible en ligne.