Rapport de Projet : Gestion des Utilisateurs avec CI/CD

1. Introduction

Le projet de gestion des utilisateurs consiste en la création d'une application permettant de gérer les utilisateurs via une interface web, tout en intégrant un pipeline CI/CD (Continuous Integration / Continuous Deployment) pour faciliter le déploiement automatisé. L'application est constituée d'un front-end en React et d'un back-end en Express, avec une base de données SQLite pour stocker les utilisateurs.

2. Objectifs du Projet

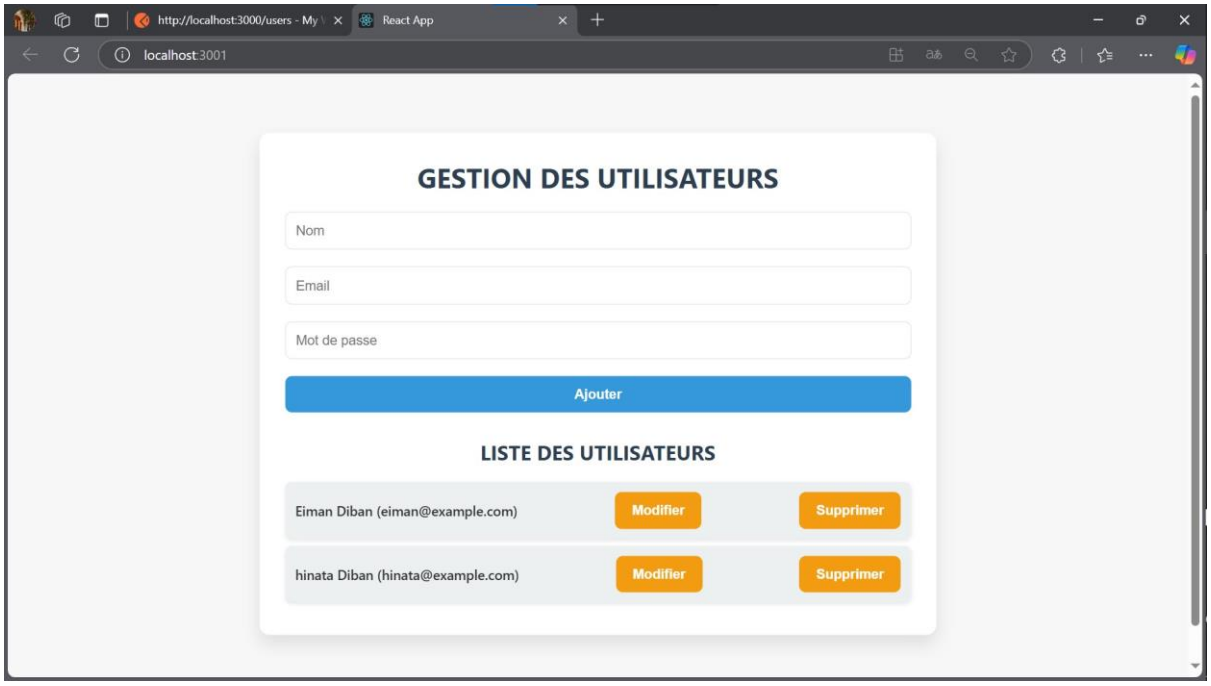
- Développer une interface utilisateur permettant la création, la modification et la suppression des utilisateurs.
- Intégrer un pipeline CI/CD pour faciliter les déploiements continus.
- Utiliser Docker pour containeriser l'application et faciliter la gestion des environnements.
- Tester l'API et l'interface utilisateur à l'aide de tests unitaires et d'intégration.

3. Description Technique

3.1. Front-End (React)

L'application front-end est développée avec React, et elle permet de gérer la liste des utilisateurs et d'afficher leurs détails. L'interface comprend plusieurs composants :

- **UserList** : Affiche la liste des utilisateurs.
- **UserForm** : Permet de créer ou de modifier un utilisateur.
- **UserDetail** : Affiche les détails d'un utilisateur spécifique.

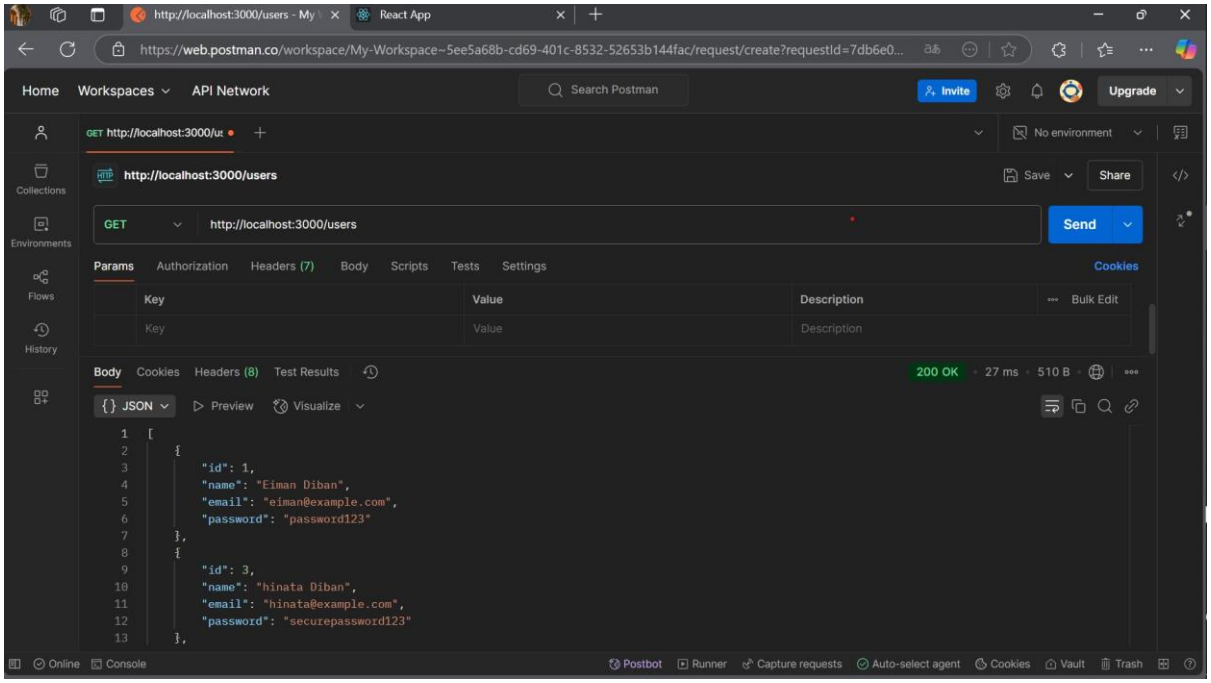


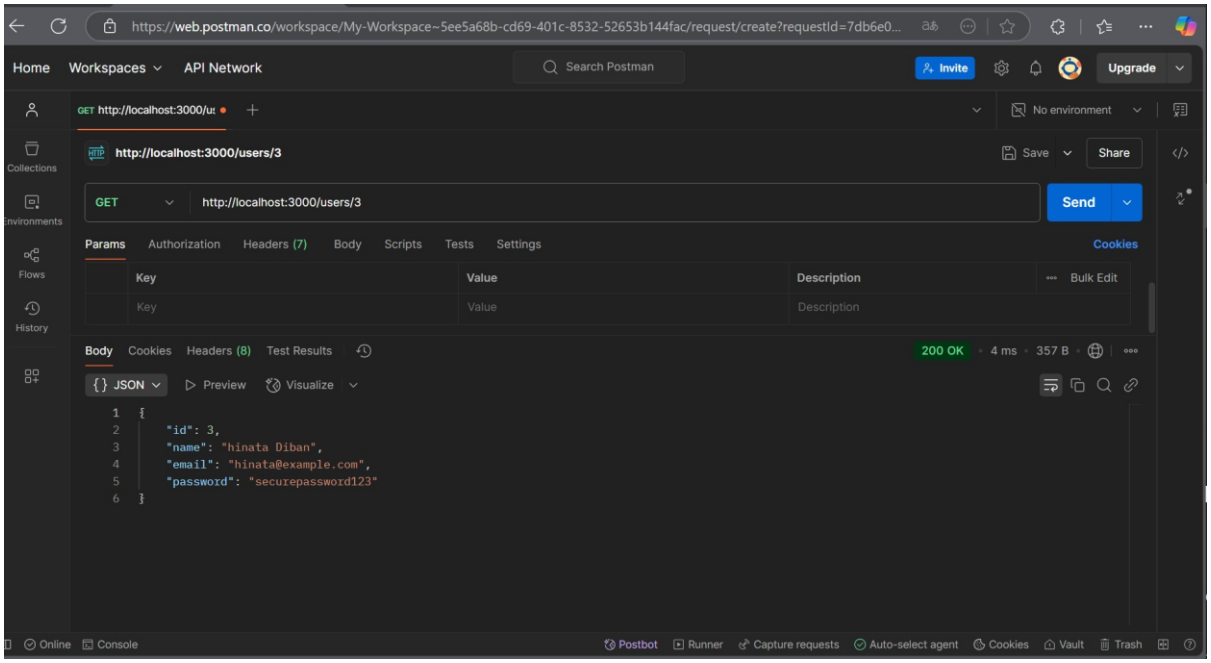
3.2. Back-End (Express)

Le back-end est développé en utilisant Express avec une base de données SQLite pour gérer les utilisateurs. Les principales routes de l'API sont :

- `GET /users` : Récupère la liste de tous les utilisateurs.
- `GET /users/:id` : Récupère un utilisateur spécifique.
- `POST /users` : Crée un nouvel utilisateur.
- `PUT /users/:id` : Modifie un utilisateur existant.
- `DELETE /users/:id` : Supprime un utilisateur.

Le serveur écoute sur le port 3001.

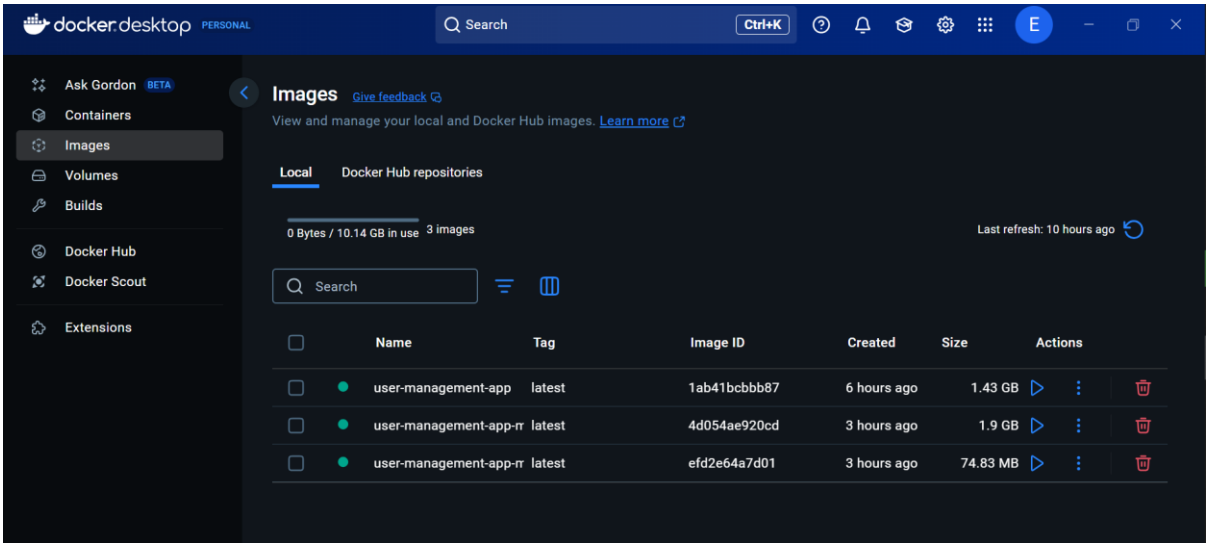




3.3. Pipeline CI/CD

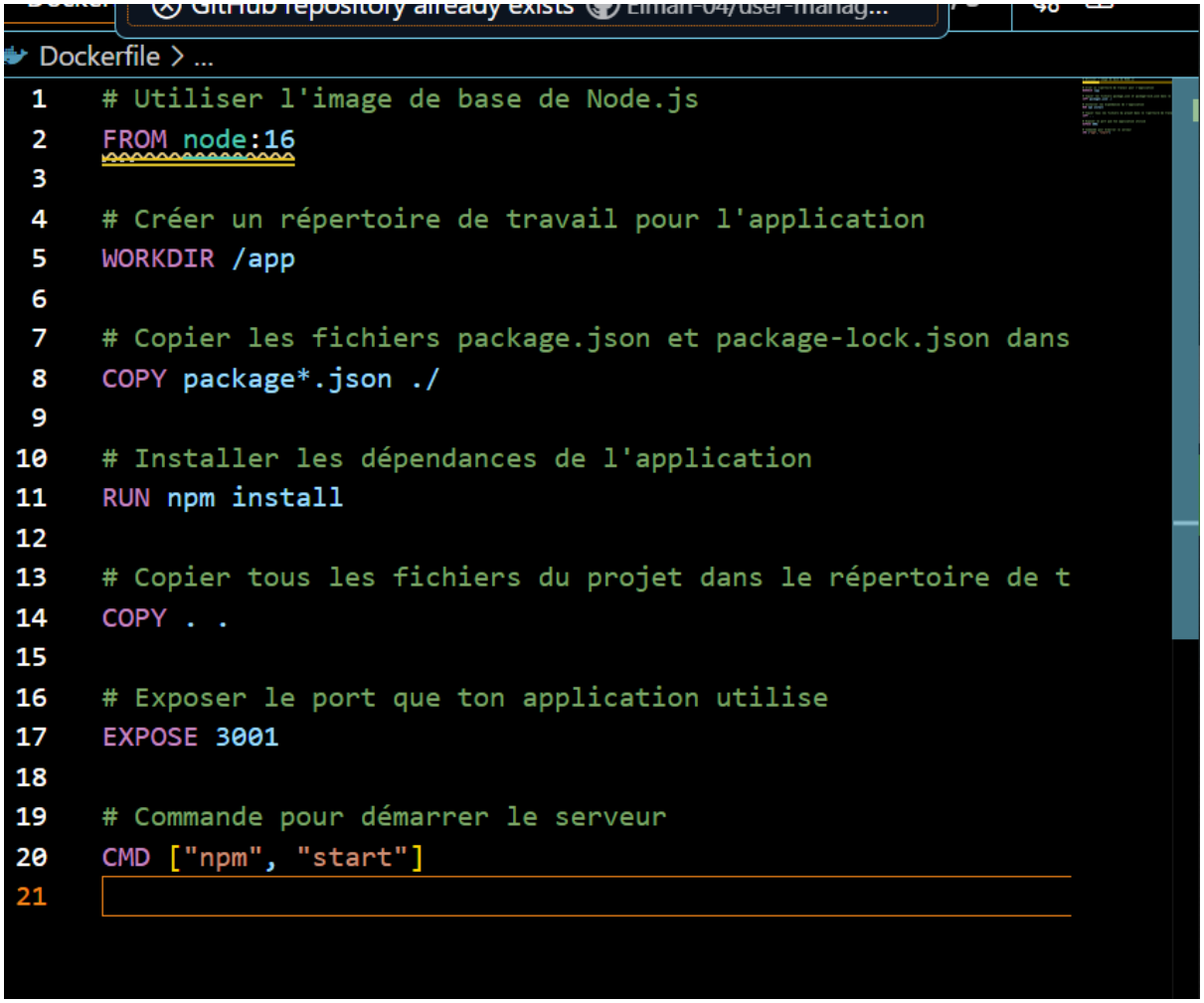
Le pipeline CI/CD est configuré dans GitHub Actions pour automatiser les tests et le déploiement de l'application. Les étapes du pipeline incluent :

- **Installation des dépendances** : Installation des dépendances Node.js.
- **Exécution des tests** : Lancement des tests unitaires pour vérifier le bon fonctionnement de l'application.
- **Build Docker** : Construction de l'image Docker de l'application.
- **Push Docker** : Envoi de l'image Docker vers Docker Hub.



3.4. Docker

Les services sont conteneurisés à l'aide de Docker. Un fichier `Dockerfile` est utilisé pour construire les images Docker pour l'application. Voici un exemple de `Dockerfile` pour le back-end :



Le front-end est également containerisé avec un `Dockerfile.frontend` qui utilise Nginx pour servir l'application React :

Dockerfile.frontend > ...

```dockerfile
# Dockerfile.frontend

# Étape 1 : Build de l'app React
FROM node:18-alpine as build
WORKDIR /app
COPY frontend/package*.json ./
RUN npm install
COPY frontend .
RUN npm run build

# Étape 2 : Serveur Nginx
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Tests

4.1. Tests Unitaires

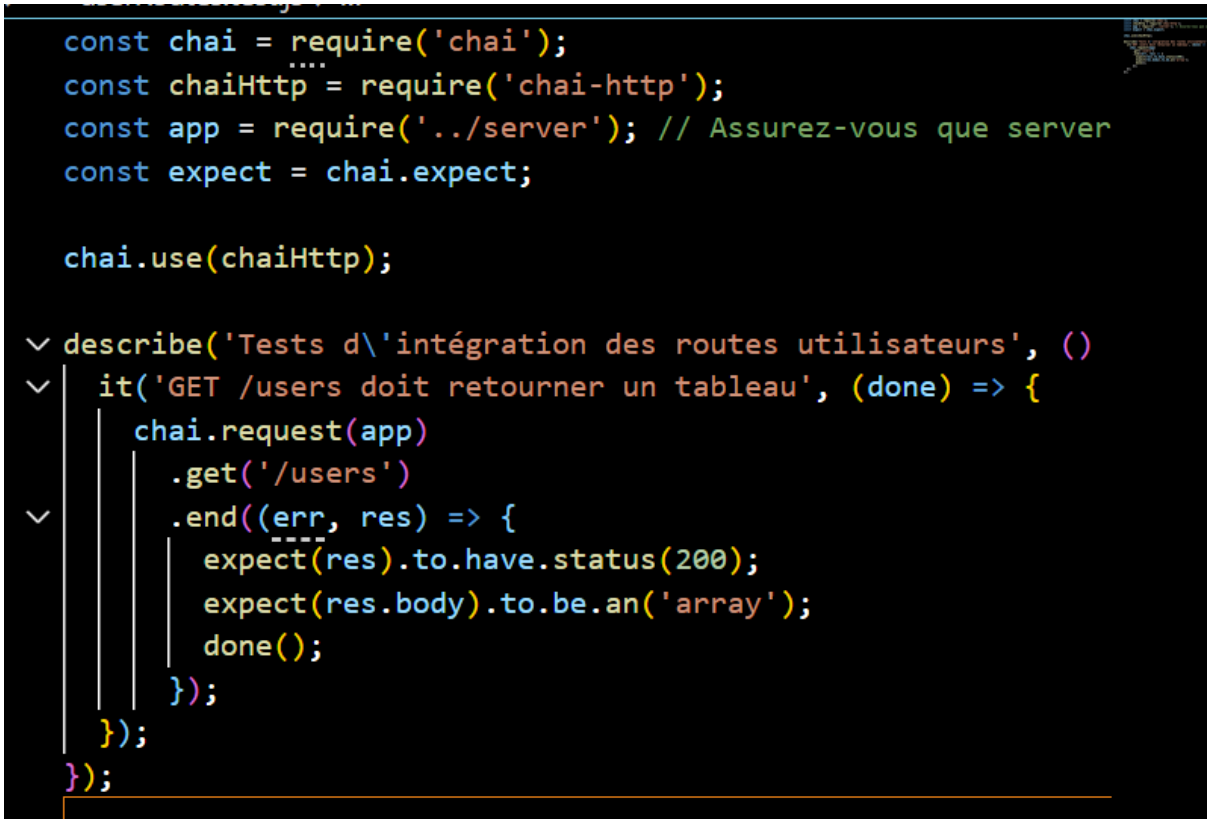
Des tests unitaires ont été écrits pour vérifier la logique de l'application. Par exemple, un test simple pour vérifier l'addition de deux nombres :

```javascript
// test/user.test.js
const chai = require('chai');
const expect = chai.expect;

describe('Exemple de test unitaire', () => {
  it('devrait retourner vrai si 2 + 2 = 4', () => {
    const result = 2 + 2;
    expect(result).to.equal(4);
  });
});
```

4.2. Tests d'Intégration

Des tests d'intégration ont également été mis en place pour tester les routes de l'API. Par exemple, un test pour vérifier que la route `GET /users` renvoie un tableau d'utilisateurs :



```
const chai = require('chai');
const chaiHttp = require('chai-http');
const app = require('../server'); // Assurez-vous que server
const expect = chai.expect;

chai.use(chaiHttp);

describe('Tests d\'intégration des routes utilisateurs', ()
  it('GET /users doit retourner un tableau', (done) => {
    chai.request(app)
      .get('/users')
      .end((err, res) => {
        expect(res).to.have.status(200);
        expect(res.body).to.be.an('array');
        done();
      });
  });
});
```

Conclusion

Le projet de gestion des utilisateurs avec CI/CD a été développé avec succès en utilisant React pour le front-end, Express pour le back-end, SQLite pour la base de données et Docker pour le déploiement. Le pipeline CI/CD mis en place avec GitHub Actions garantit que l'application est automatiquement testée et déployée à chaque modification du code. Les tests unitaires et d'intégration assurent que l'application fonctionne correctement avant le déploiement en production.