

Rapport de présentation du projet : Application Fullstack de gestion des utilisateurs



par
Razzouk Majda
LP IAWM

SOMMAIRE

1.Introduction

1.1 Objectifs du projet

1.2 Technologies utilisées

1.3 Architecture globale

2. Développement de l'application

2.1 Backend avec Express.js

2.2 Frontend avec React.js

2.3 Communication API (Axios)

2.4 Tests des API avec Postman

3. Dockerisation de l'application

4. Mise en œuvre des tests

5. Intégration Continue avec GitHub Actions

6.Déploiement de l'application

7. Conclusion

1. Introduction

1.1 Objectifs du projet

L'objectif principal de ce projet est la création d'une application web full-stack dédiée à la gestion des utilisateurs. Elle propose une interface intuitive permettant d'effectuer des opérations CRUD à travers une API REST conçue avec Express.js. Côté client, React.js assure une interface moderne et responsive.

Le projet intègre également des pratiques professionnelles telles que la dockerisation, les tests automatisés avec Mocha/Chai et l'intégration continue via GitHub Actions. Il s'agit aussi d'un premier usage de PostgreSQL comme système de gestion de base de données.

1.2 Technologies utilisées

Backend :

- **Node.js, Express.js**
- **PostgreSQL avec le client **pg****
- **Mocha, Chai, Supertest pour les tests**

● **Frontend :**

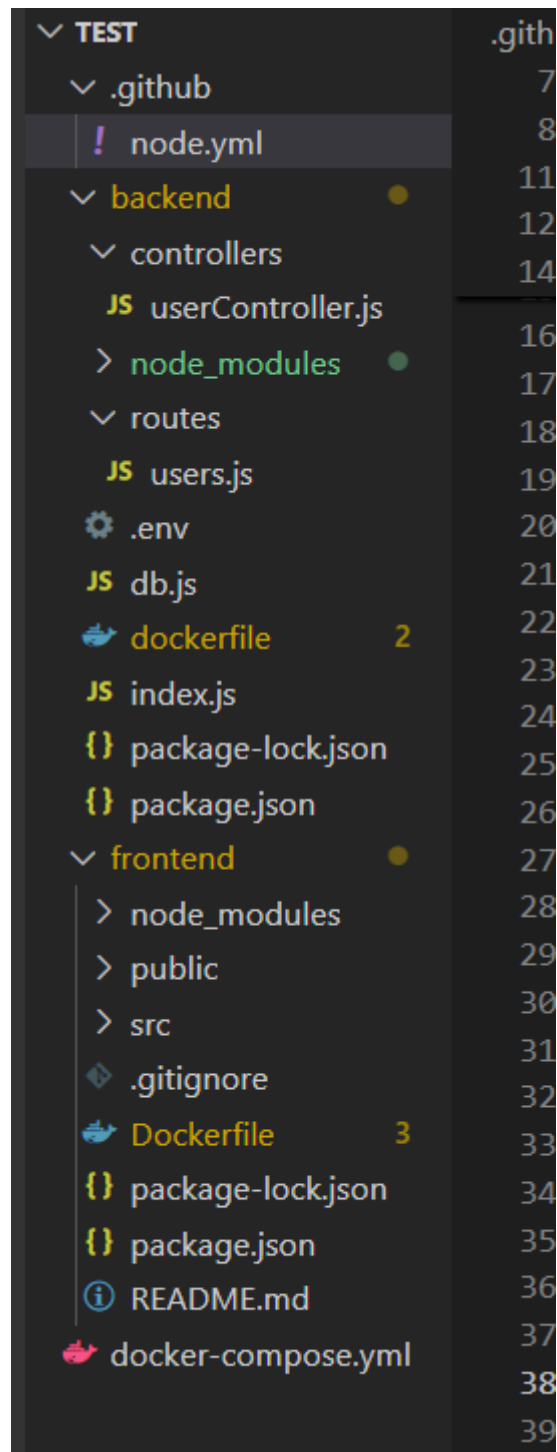
- **React.js via Vite**
- **Axios**
- **CSS**

● **DevOps / Infrastructure :**

- **Docker & Docker Compose**
- **Git, GitHub**
- **GitHub Actions**
- **Nginx (serveur de production)**

1.3 Architecture globale

L'application repose sur une architecture en trois services : le frontend (React), le backend (Express), et PostgreSQL, le tout conteneurisé avec Docker et orchestré via Docker Compose. L'organisation du projet suit une structure claire avec des dossiers dédiés et un `docker-compose.yml` à la racine.



2. Développement de l'application

2.1 Backend avec Express.js

Le backend offre une API REST pour gérer les utilisateurs. Les principales routes permettent de :

- Lister tous les utilisateurs
- Récupérer un utilisateur par ID
- Ajouter, modifier et supprimer un utilisateur



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like .github, node_modules, backend, controllers, routes, and frontend. The code editor shows the file users.js in the routes folder, which contains the following code:

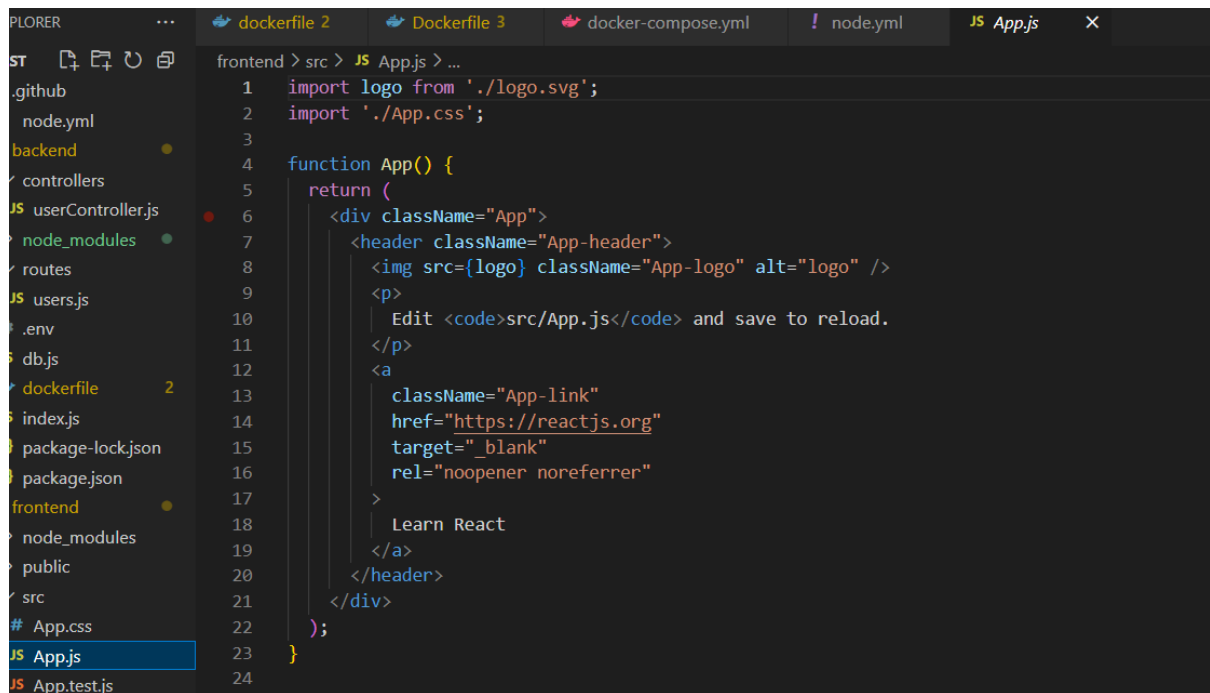
```
1 const express = require('express');
2 const router = express.Router();
3 const {
4   getUsers,
5   addUser,
6   updateUser,
7   deleteUser
8 } = require('../controllers/userController');
9
10 router.get('/', getUsers);
11 router.post('/', addUser);
12 router.put('/:id', updateUser);
13 router.delete('/:id', deleteUser);
14
15 module.exports = router;
```

La base de données PostgreSQL gère la persistance des données. L'environnement est simple à configurer et optimisé pour les opérations CRUD.

2.2 Frontend avec React.js

L'interface utilisateur développée avec React.js permet d'interagir facilement avec le backend :

- Affichage et gestion de la liste des utilisateurs
- Formulaire pour l'ajout ou l'édition
- Suppression d'utilisateurs
-



```

1  import logo from './logo.svg';
2  import './App.css';
3
4  function App() {
5    return (
6      <div className="App">
7        <header className="App-header">
8          <img src={logo} className="App-logo" alt="logo" />
9          <p>
10             Edit <code>src/App.js</code> and save to reload.
11          </p>
12          <a
13             className="App-link"
14             href="https://reactjs.org"
15             target="_blank"
16             rel="noopener noreferrer"
17          >
18             Learn React
19          </a>
20        </header>
21      </div>
22    );
23  }
24

```

Le tout fonctionne avec Vite, pour un démarrage rapide.

2.3 Communication API (Axios)

La communication entre les deux couches est assurée par Axios. Voici quelques exemples :

- `axios.get("/users")` : liste des utilisateurs
- `axios.post("/users", data)` : ajout
- `axios.put("/users/:id", data)` : modification
- `axios.delete("/users/:id")` : suppression

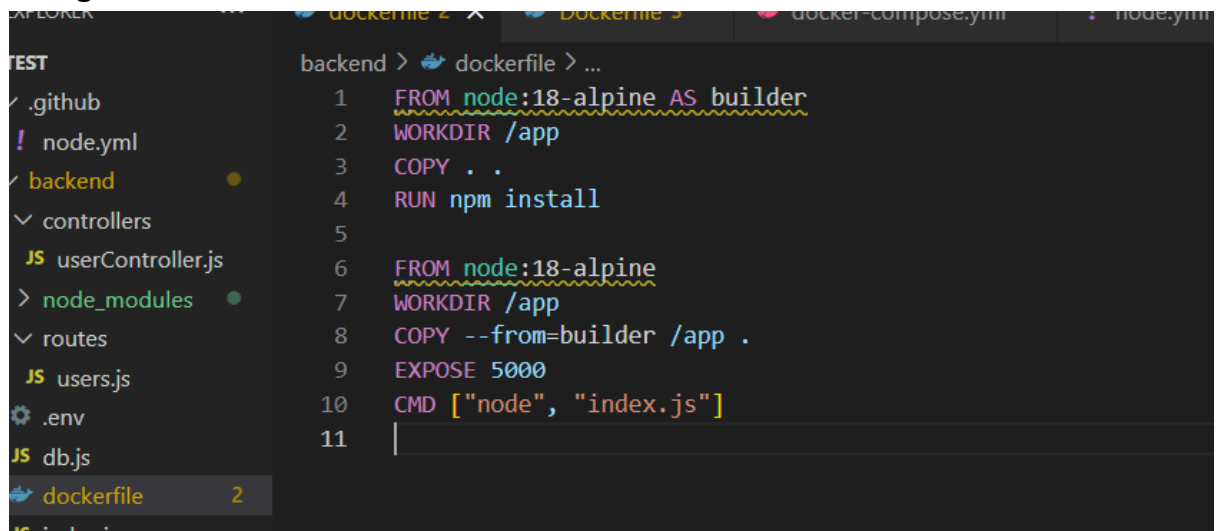
2.4 Tests des API avec Postman

Chaque route a été testée via Postman pour valider son fonctionnement. Des captures d'écran illustrent les résultats des requêtes GET, POST, PUT et DELETE, avec vérification via pgAdmin.

3. Dockerisation de l'application

Trois éléments ont été dockerisés :

- Backend (Node.js) avec Dockerfile dédié
- Frontend (React.js) construit puis servi par Nginx
- PostgreSQL



```
backend > dockerfile > ...
1 FROM node:18-alpine AS builder
2 WORKDIR /app
3 COPY . .
4 RUN npm install
5
6 FROM node:18-alpine
7 WORKDIR /app
8 COPY --from=builder /app .
9 EXPOSE 5000
10 CMD ["node", "index.js"]
11
```

Le tout est orchestré avec `docker-compose.yml`, permettant une installation complète avec une seule commande.

4. Mise en œuvre des tests

- Tests unitaires : pour chaque route (ex. `GET /users`), vérification que la réponse est correcte.
- Tests d'intégration : exécution d'un `POST /users`, vérifiant la bonne insertion dans la base.

- Couverture de code : via NYC pour visualiser les portions couvertes par les tests.
- Scripts de test : définis dans `package.json` pour simplifier l'exécution locale.

Un bug est apparu lors du test `GET /users` (TypeError). Des pistes d'amélioration ont été proposées, notamment la correction de l'import `chai.request` et le renforcement des tests.

5. Intégration Continue avec GitHub Actions

Un pipeline CI/CD a été défini via GitHub Actions pour :

- Vérifier le code à chaque push
- Lancer automatiquement les tests
- Construire et publier l'image Docker
- Déployer le projet

Le fichier `.yaml` est placé dans `.github/workflows/`.

Un problème de permissions a empêché l'exécution de `mocha`, mettant en évidence la nécessité d'une meilleure configuration des droits d'exécution.

6. Déploiement de l'application

- Frontend : déployé sur [Vercel](#), avec mise à jour automatique à chaque push.
- Backend : non supporté par Vercel, nécessite une alternative comme un VPS.

7. Conclusion

Ce projet a été une belle opportunité pour explorer toutes les étapes d'un développement full-stack moderne. Il a permis :

- La maîtrise de React.js, Express.js et PostgreSQL
- L'intégration des bonnes pratiques DevOps (Docker, CI/CD)
- La mise en œuvre de tests automatisés

7.1 Bilan

Une expérience complète, alliant technique, logique, et résolution de problèmes.

7.2 Améliorations futures

- Résoudre les erreurs des tests
- Ajouter une authentification (JWT)
- Créer un dashboard d'administration pour le suivi statistique