Développement Fullstack + CI/CD avec GitHub Actions

Réalisé par : Belasri safia IAWM

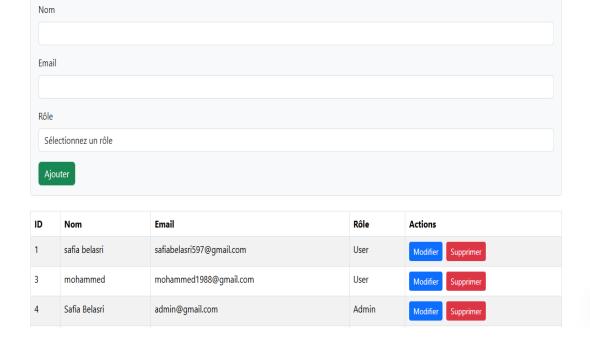
Présentation Générale du Projet

Objectif du projet

Ce projet a pour objectif de développer une application **full stack** de **gestion des utilisateurs**. L'application permet d'ajouter, afficher, modifier et supprimer des utilisateurs via une interface web moderne, avec un backend RESTful API. Elle a également été **conteneurisée avec Docker** et **déployée via GitHub Actions** pour mettre en place un pipeline CI/CD.

Les pages:





Technologies utilisées

Frontend (client)

- **React.js**: Librairie JavaScript pour construire une interface utilisateur dynamique.
- Axios: Pour consommer l'API backend.

Backend (serveur)

- **Express.js**: Framework Node.js pour créer l'API REST.
- **SQLite**: Base de données légère, embarquée dans le backend.

Tests

• Mocha, Chai: Pour les tests unitaires et d'intégration (backend).

Conteneurisation & Déploiement

- Docker / Docker Compose : Pour conteneuriser le frontend, backend et base de données.
- **GitHub Actions**: Pour automatiser les étapes CI/CD (build, test, push image Docker).
- **Docker Hub** : Pour héberger l'image Docker de l'application.

Fonctionnalités principales

- Ajout d'un utilisateur avec nom, email, et photo.
- Liste des utilisateurs avec visualisation des informations.
- Modification et suppression d'un utilisateur.
- Tests unitaires sur les Endpoint de l'API.
- CI/CD complet avec GitHub Actions.
- Déploiement sur Docker Hub

Étapes de mise en place du Backend et du Frontend

Ce projet repose sur une architecture **full stack JavaScript**, avec un **backend Express.js** relié à une base de données **SQLite** et un **frontend React** consommant l'API
REST. Cette section décrit les étapes techniques mises en œuvre pour installer et configurer ces deux couches.

Mise en place du Backend (Express.js + SQLite)

Structure du dossier /backend

- server.js: point d'entrée de l'application et gestion de la base de données SQLite et routes de l'API REST (GET, POST, PUT, DELETE).
- logique métier des utilisateurs.
- définition du modèle de données.
- test/api.test.js: tests unitaires et d'intégration avec chai.
- package.json: configuration du projet Node.js.

Étapes de mise en place

1. Initialisation du projet

mkdir user_managment

cd user_managment

<mark>npm init -y</mark>

2. Installation des dépendances

npm install express sqlite3 cors body-parser

- 3. Développement du backend
 - Connexion à la base SQLite avec sqlite3.
 - o Création des routes REST pour gérer les utilisateurs (CRUD).
- 4. Lancement du serveur

node server.js

Le serveur est accessible sur http://localhost:5000.

Mise en place du Frontend (React)

Structure du dossier /frontend

- src/App.js : structure principale de l'application et : formulaire d'ajout et d'édition et affichage de la liste des utilisateurs.
- src/index.js : point d'entrée de React.

Étapes de mise en place

1. Création de l'application React

npx create-react-app frontend

cd frontend

2. Installation d'Axios pour consommer l'API

npm install axios

3. Développement des composants

- o App.js: effectue un appel GET pour afficher les utilisateurs.
- o APP.js: envoie des requêtes POST/PUT/DELETE vers l'API.
- 4. **Connexion avec l'API** Les appels Axios sont configurés pour pointer vers http://localhost:5000/users.
- 5. Lancement de l'interface

npm start

L'application React est disponible sur http://localhost:3000.

Explication de la base de données

Pour ce projet, nous avons utilisé une base de données **SQLite**, un moteur de base de données léger, embarqué et facile à configurer.

Pourquoi SQLite?

- Pas besoin de serveur de base de données externe.
- Adapté aux projets de petite à moyenne envergure.
- Compatible avec Docker (pas de services à configurer séparément).

Structure de la table users :

Voici les colonnes définies dans notre base :

Champ	Туре	Description	
id	INTEGER	Identifiant unique (clé primaire)	
Nom	TEXT	Nom de l'utilisateur	
Email	TEXT	Email personnel	
Rôle	TEXT	Choisi le rôle admin ou user	

Initialisation

La base est automatiquement initialisée si elle n'existe pas. Cela se fait via un fichier server.js exécuté au lancement du serveur.

Documentation de l'API:

Méthode	Endpoint	Description	Corps attendu (JSON)	Réponse HTTP
GET	/users	Récupère tous les utilisateurs	_	200 OK
POST	/users	Ajoute un nouvel utilisateur	{id: 4,name: 'Safia Belasri',email: 'admin@gmail.com',rôle: 'Admin'},	201 Created
PUT	/users/:id	Modifie un utilisateur existant	{id: 4,name: 'Safia Belasri',email: 'admin@gmail.com',rôle: 'Admin'},	200 OK
DELETE	/users/:id	Supprime un utilisateur		200 OK

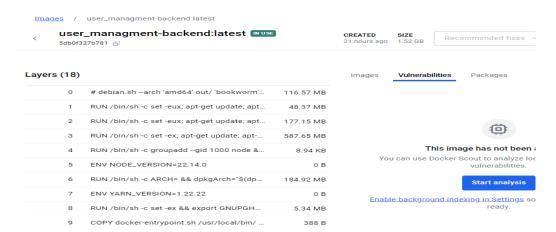
Dockerisation: étapes et choix faits

La dockerisation nous permet d'exécuter l'ensemble du projet dans des conteneurs isolés et reproductibles.

Backend - Dockerfile

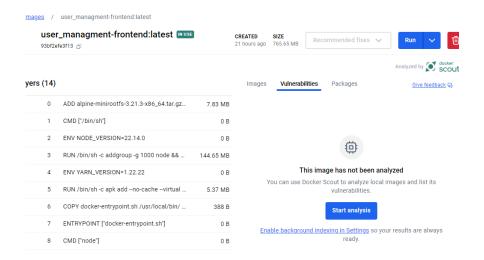
Utilisation de Node.js v22 comme base.

Lancement du serveur sur le port 5000.



Frontend - Dockerfile

- Étape 1 : build React dans une image Node.
- · Assets statiques optimisés automatiquement par npm run build.



docker-compose.yml

```
# SUPPRIMEZ la ligne version:
services:
    build: .
    backend:
    ports:
      - '5000:5000'
      - ./users.db:/app/users.db
      - ./server.js:/app/server.js
      - ./package.json:/app/package.json
      - ./package-lock.json:/app/package-lock.json
    environment:
      - NODE ENV=development
    restart: always
  frontend:
    build: ./frontend
    ports:
      - '3000:80'
    restart: always
```

- Les ports exposés sont ceux utilisés en développement.
- Le frontend dépend du backend pour démarrer.

tests, actions GitHub, conteneurs Docker

api.test.js

Ton test doit être écrit en commonJS:

Voir I dossier test/api.test.js

Lance le test:

npm test

```
at Resolver.resolvemodule (node_modules/jest-resolve/bulld/resolver.js:324:11)
at Object.<anonymous> (src/setupTests.js:5:1)
at TestScheduler.scheduleTests (node_modules/@jest/core/build/TestScheduler.js:333:13)
at runJest (node_modules/@jest/core/build/runJest.js:404:19)

Test Suites: 1 failed, 1 total
Tests: 0 total
Snapshots: 0 total
Time: 1.763 s
Ran all test suites.

Watch Usage

> Press f to run only failed tests.

> Press o to only run tests related to changed files.

> Press q to quit watch mode.

> Press p to filter by a filename regex pattern.

> Press t to filter by a test name regex pattern.

> Press Enter to trigger a test run.
```

GitHub Actions: pipeline CI/CD

Voici le fichier .github/workflows/ci.yml utilisé pour automatiser les étapes CI/CD.

Difficultés rencontrées et solutions

1. Conflit de ports Docker

- **Problème :** Le port 5000 était déjà utilisé sur le système hôte, empêchant le conteneur backend de démarrer.
- **Solution :** Modification du fichier docker-compose.yml pour mapper le port 5000 du conteneur sur un autre port libre sur l'hôte, par exemple 5001:5000.

2. Erreur ELF dans le conteneur backend (sqlite3)

- **Problème :** Le module sqlite3 échouait à se charger dans le conteneur à cause d'un "invalid ELF header".
- Cause: Le module avait été installé sur la machine hôte et copié dans le conteneur. Or, il doit être compilé directement dans l'environnement Docker.
- **Solution :** Exécution de npm install directement **dans** le conteneur lors du build (dans le Dockerfile) pour éviter l'incompatibilité binaire.

3. Problème avec les tests (module chai-http)

- Problème: Erreur MODULE_NOT_FOUND ou ERR_PACKAGE_PATH_NOT_EXPORTED pour chai-http.
- Solution:
 - o Installation manuelle du module avec :

npm install --save-dev chai-http

 Si l'erreur persiste, utiliser require('chai').use(require('chai-http')) au lieu d'un import.

4. Erreurs de configuration GitHub Actions

• Problème: Mauvaise configuration des secrets ou étapes du pipeline CI/CD.

• Solution:

- Vérification des noms exacts des secrets (DOCKER_USERNAME, DOCKER_PASSWORD, etc.)
- Test du workflow en local avec act ou débogage ligne par ligne via GitHub Actions.

Conclusion

Ce projet de gestion des utilisateurs m'a permis de mettre en œuvre les compétences acquises en développement full-stack avec une approche DevOps. Grâce à l'utilisation de **React** pour le frontend et **Express.js avec SQLite** pour le backend, j'ai pu concevoir une application complète, fonctionnelle et intuitive.

La **dockerisation** des services et l'intégration d'un **pipeline CI/CD via GitHub Actions** ont permis d'automatiser les processus de test, build et déploiement, rendant l'application facilement maintenable et déployable.

Ce projet m'a également permis de faire face à des difficultés techniques (modules incompatibles, erreurs de build, configuration des workflows), que j'ai su résoudre en cherchant activement des solutions et en documentant chaque étape.

En résumé, cette expérience a été enrichissante tant sur le plan technique que méthodologique, en me préparant concrètement aux exigences des projets en environnement professionnel.

.