

Application CRUD Utilisateurs






Développement Full-Stack avec React, Express.js, MySQL et Docker

1. Présentation Générale du Projet

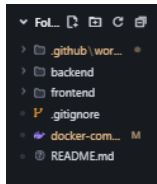
Objectif :

Développer une application web complète permettant de gérer les utilisateurs via des opérations CRUD (Create, Read, Update, Delete).

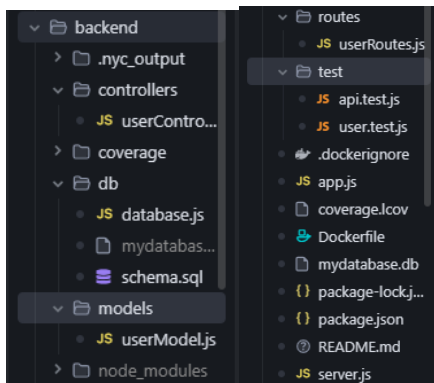
Technologies utilisées :

-  Frontend : React.js
-  Backend : Express.js (Node.js)
-  Base de données : MySQL
-  Conteneurisation : Docker + Docker Compose
-  CI/CD : GitHub Actions

2. Mise en Place du Projet

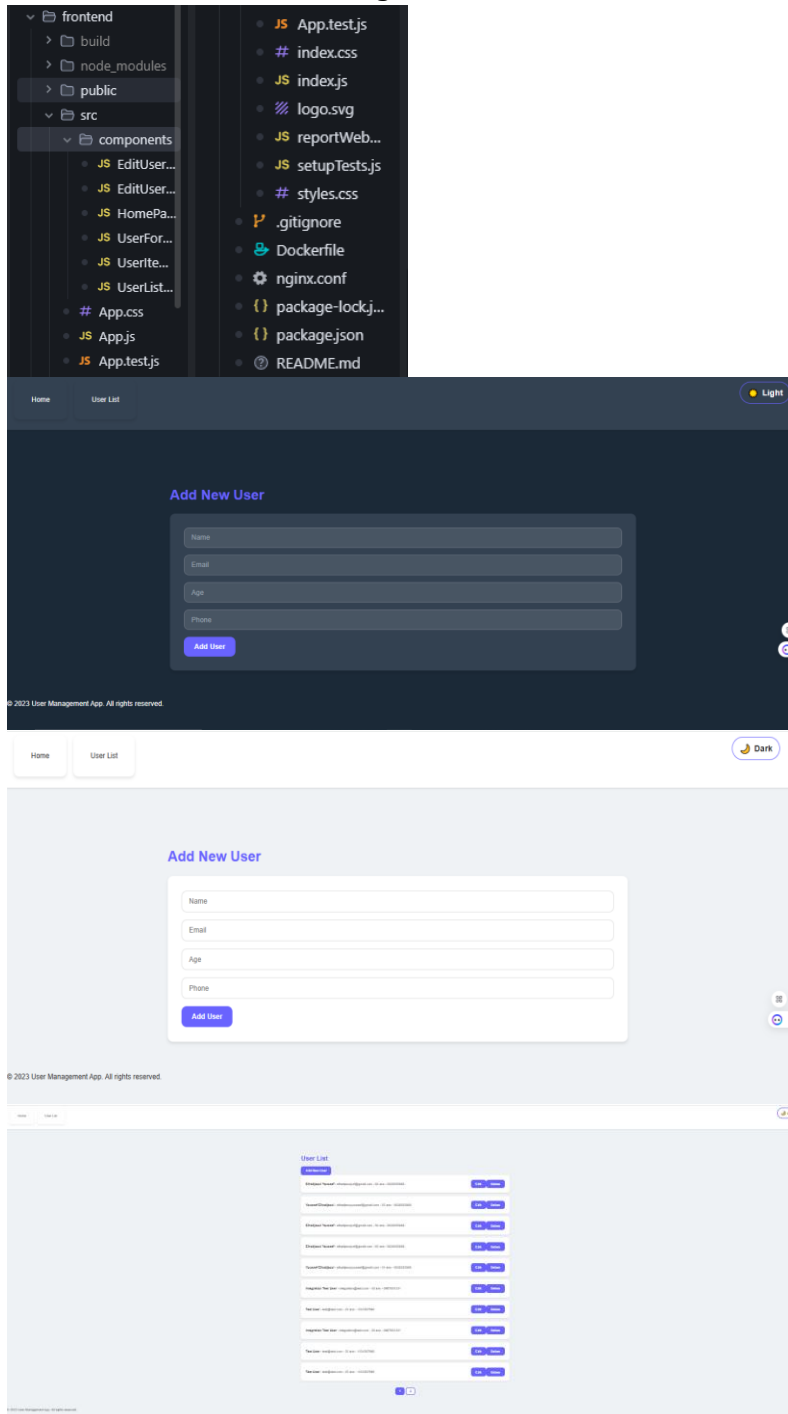


2.1 Backend – Express.js



2.2 Frontend – React.js

- Interface utilisateur responsive et intuitive
- Utilisation du useState, useEffect et axios pour interagir avec l'API
- Système de routing avec react-router-dom
- Validation des formulaires et gestion d'erreurs côté client



3. Base de Données – MySQL

- Conception du schéma avec table users (id, nom, email, etc.)
- Mise en place des contraintes d'intégrité (clés primaires, uniques, etc.)
- Connexion sécurisée avec le backend via mysql2



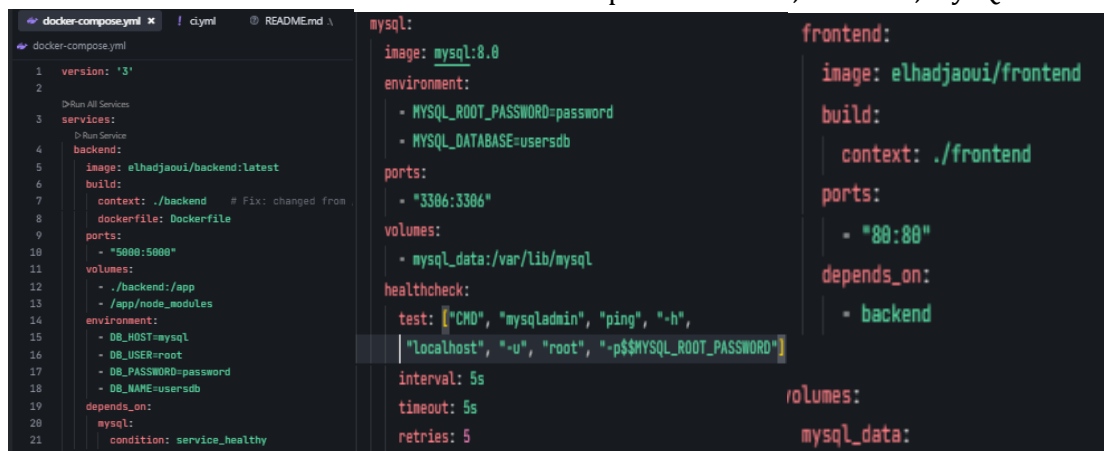
The screenshot shows two files in a VS Code editor. The top file, `schema.sql`, contains a SQL statement to create a table named `users` with columns `id` (primary key), `name`, `email`, `age`, and `phone`. The bottom file, `database.js`, shows the Node.js code for connecting to a MySQL database using the `mysql2` library. It uses environment variables for database configuration and includes error handling for connection failures.

```
backend / db / schema.sql
1 CREATE TABLE IF NOT EXISTS users (
2   id INT AUTO_INCREMENT PRIMARY KEY,
3   name VARCHAR(255) NOT NULL,
4   email VARCHAR(255) NOT NULL,
5   age INT,
6   phone VARCHAR(20)
7 );

backend / db / JS database.js / ...
1 const mysql = require('mysql2');
2
3 const db = mysql.createConnection({
4   host: process.env.DB_HOST || 'localhost',
5   user: process.env.DB_USER || 'root',
6   password: process.env.DB_PASSWORD || 'password',
7   database: process.env.DB_NAME || 'usersdb',
8   port: process.env.DB_PORT || 3306
9 });
10
11 // Handle connection errors
12 db.connect((err) => {
13   if (err) {
14     console.error('Database connection failed:', err);
15     return;
16   }
17   console.log('Connected to database');
18 });
19
20 module.exports = db;
```

4. Dockerisation

- Architecture multi-conteneurs avec docker-compose : backend, frontend, MySQL



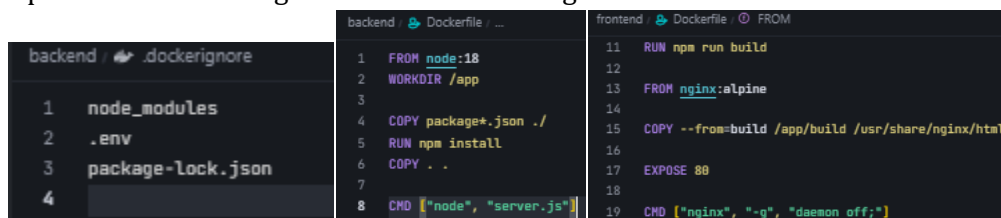
The screenshot shows the `docker-compose.yml` file in VS Code. It defines three services: `mysql`, `backend`, and `frontend`. The `mysql` service uses the `mysql:8.0` image and sets environment variables for root password and database name. The `backend` service uses the `elhadjaoui/backend:latest` image and sets environment variables for database connection. The `frontend` service uses the `elhadjaoui/frontend` image and sets the build context to `./frontend`. The `backend` service depends on the `mysql` service.

```
docker-compose.yml
1 version: '3'
2
3 services:
4   backend:
5     image: elhadjaoui/backend:latest
6     build:
7       context: ./backend # Fix: changed from
8       dockerfile: Dockerfile
9     ports:
10      - "5000:5000"
11     volumes:
12      - ./backend:/app
13      - /app/node_modules
14     environment:
15      - DB_HOST=mysql
16      - DB_USER=root
17      - DB_PASSWORD=password
18      - DB_NAME=usersdb
19     depends_on:
20      mysql:
21        condition: service_healthy

mysql:
  image: mysql:8.0
  environment:
    - MYSQL_ROOT_PASSWORD=password
    - MYSQL_DATABASE=usersdb
  ports:
    - "3306:3306"
  volumes:
    - mysql_data:/var/lib/mysql
  healthcheck:
    test: ["CMD", "mysqladmin", "ping", "-h",
           "localhost", "-u", "root", "-p${MYSQL_ROOT_PASSWORD}"]
    interval: 5s
    timeout: 5s
    retries: 5

frontend:
  image: elhadjaoui/frontend
  build:
    context: ./frontend
  ports:
    - "80:80"
  depends_on:
    - backend
  volumes:
    mysql_data:
```

- Optimisation des images Docker via `.dockerignore` et `Dockerfile`



The screenshot shows two files in a VS Code editor. The left file, `.dockerignore`, lists `node_modules`, `.env`, and `package-lock.json` to be ignored. The right file, `Dockerfile`, shows the build process for the frontend service, including copying the package manager files, installing dependencies, and exposing port 80.

```
backend / .dockerignore
1 node_modules
2 .env
3 package-lock.json
4

backend / Dockerfile / ...
1 FROM node:18
2 WORKDIR /app
3
4 COPY package*.json ./
5 RUN npm install
6 COPY . .
7
8 CMD ["node", "server.js"]

frontend / Dockerfile / FROM
11 RUN npm run build
12
13 FROM nginx:alpine
14
15 COPY --from=build /app/build /usr/share/nginx/html
16
17 EXPOSE 80
18
19 CMD ["nginx", "-g", "daemon off;"]
```

- Montage de volumes pour la persistance des données
- Configuration d'un réseau privé Docker pour la communication inter-conteneurs

5. Intégration Continue / Déploiement Continu (CI/CD)

- Mise en place de workflows GitHub Actions :
- Tests automatisés à chaque push
- Build des images Docker
- Push automatique vers Docker Hub

The image shows a GitHub Actions workflow file named `ci.yml` in the `.github/workflows` directory. The workflow is titled "CI/CD Pipeline" and is triggered on pushes to the `main` branch. It consists of several jobs: `test-and-build`, `Set up Docker Buildx`, and `Build and Push Docker Images`. The `test-and-build` job runs on `ubuntu-latest` and includes a service for `mysql:8.0` with environment variables for root password and database name, and a health check. The `Set up Docker Buildx` job sets up the Docker Buildx environment. The `Build and Push Docker Images` job builds and pushes Docker images for the `backend` and `frontend` directories to Docker Hub.

```

1 name: CI/CD Pipeline
2
3 on:
4   push:
5     branches: [ main ]
6   pull_request:
7     branches: [ main ]
8
9 jobs:
10  test-and-build:
11    runs-on: ubuntu-latest
12
13    services:
14      mysql:
15        image: mysql:8.0
16        env:
17          MYSQL_ROOT_PASSWORD: password
18          MYSQL_DATABASE: usersdb
19        ports:
20          - 3306:3306
21        options: --health-cmd="mysqladmin ping"
22                --health-interval=10s
23                --health-timeout=5s
24                --health-retries=3
25
26  - name: Set up Docker Buildx
27    uses: docker/setup-buildx-action@v1
28
29  - name: Build and Push Docker Images
30    env:
31      DOCKER_USERNAME: ${ secrets.DOCKER_USERNAME }
32      DOCKER_PASSWORD: ${ secrets.DOCKER_PASSWORD }
33    run: |
34      echo $DOCKER_PASSWORD | docker login -u $DOCKER_USERNAME --password-stdin
35      docker build -t elhadjaoui/backend:latest ./backend
36      docker build -t elhadjaoui/frontend:latest ./frontend
37      docker push elhadjaoui/backend:latest
38      docker push elhadjaoui/frontend:latest

```

The screenshot also shows the Docker Hub repository structure with folders for `backend` and `frontend`, and a `README.md` file.

6. Tests et Validation

- Tests de toutes les routes API (GET, POST, PUT, DELETE) via Postman
- Vérification manuelle de l'interface utilisateur
- Contrôle de la communication entre frontend ↔ backend ↔ base de données

GEThttp://localhost:5000/api/users

DELETEhttp://localhost:5000/api/users/5

ParamsAuthorizationHeaders (6)BodyScriptsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQL

This request does not have a body.

BodyCookiesHeaders (8)Test Results200 OK

JSONPreviewVisualize

```
1 {
2   {
3     "id": 2,
4     "name": "Elhadjaoui Youssef",
5     "email": "elhadjaouiysf@gmail.com",
6     "age": 66,
7     "phone": "0628205868"
8   },
9   {
10    "id": 4,
11    "name": "Youssef Elhadjaoui",
12    "email": "elhadjaouiyoussyf@gmail.com",
13    "age": 65,
14    "phone": "0628205868"
15  },
16  {
17    "id": 6,
18    "name": "Elhadjaoui Youssef",
19    "age": 44,
20    "phone": "56566565"
21  }
22 }
```

Query Params

Key	Value
Key	Value

BodyCookiesHeaders (8)Test Results200 OK

JSONPreviewVisualize

```
1 {
2   "message": "User deleted successfully"
3 }
```

User CRUD API / Create User

POSThttp://localhost:5000/api/users

ParamsAuthorizationHeaders (8)BodyScriptsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQL

```
1 {
2   "name": "Youssef Elhadjaoui",
3   "email": "elhadjaouiysf@gmail.com",
4   "age": 22,
5   "phone": "0628205868"
6 }
```

BodyCookiesHeaders (8)Test Results200 OK

JSONPreviewVisualize

```
1 {
2   "id": 27,
3   "name": "Youssef Elhadjaoui",
4   "email": "elhadjaouiysf@gmail.com",
5   "age": 22,
6   "phone": "0628205868"
7 }
```

User CRUD API / Update User

PUThttp://localhost:5000/api/users/6

ParamsAuthorizationHeaders (8)BodyScriptsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQL

```
1 {
2   "name": "wac",
3   "email": "wac@gamil.com",
4   "age": 44,
5   "phone": "56566565"
6 }
```

BodyCookiesHeaders (8)Test Results200 OK

JSONPreviewVisualize

```
1 {
2   "id": "6",
3   "name": "wac",
4   "email": "wac@gamil.com",
5   "age": 44,
6   "phone": "56566565"
7 }
```

PS E:\IAWM\deploiment\user-crud-app\backend> \$env:DB_HOST="localhost"; \$env:DB_PORT="3386"; npm test

> backend@1.0.0 test

> nyc mocha --exit

Connected to database

API Integration Tests

✓ should get all users (111ms)

✓ should create a new user (244ms)

✓ should get a user by id

✓ should update a user (174ms)

✓ should delete a user (187ms)

✓ should handle invalid user creation

✓ should handle getting non-existent user

✓ should handle invalid update data

✓ should handle deleting non-existent user (53ms)

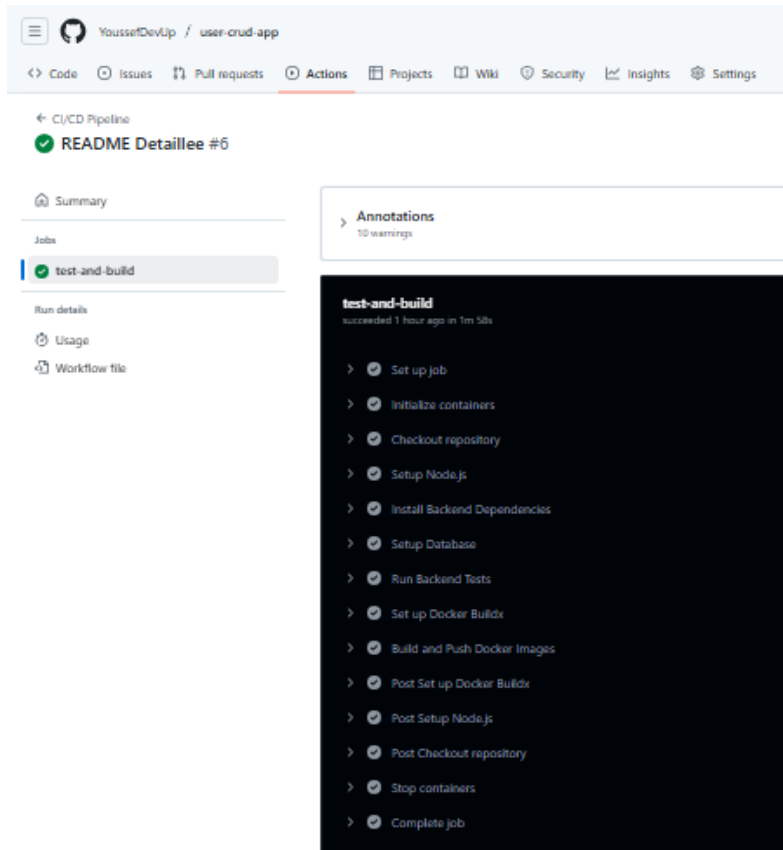
User Model Tests

✓ should create a new user

✓ should get all users

11 passing (6s)

% Stats	% Branch	% Funcs	% Lines	Uncovered Lines
84.09	75	94.73	89.15	
76.92	50	0	76.92	
76.92	50	0	76.92	12-14
84.09	78.57	100	94.87	
84.09	78.57	100	94.87	52-53
75	75	100	75	
75	75	100	75	14-15
85.71	50	100	85.71	
100	100	100		



7. Difficultés Rencontrées et Solutions

- Connexion refusée entre services : Configuration des ports et des réseaux Docker
- Erreurs CORS : Ajout du middleware cors() dans Express
- Synchronisation base de données : Utilisation de wait-for-it.sh ou dépendances dans Docker
- Communication frontend/backend : Utilisation de variables d'environnement pour les URL d'API
- Leçons apprises : Importance de bien gérer la communication entre conteneurs, structuration claire du projet, CI/CD

8. Conclusion et Pistes d'Amélioration

- Objectifs atteints : Développement et déploiement réussi d'une application CRUD full-stack, conteneurisée, testée, et intégrée via CI/CD.
- Axes d'amélioration : Ajout de l'authentification (JWT), tests end-to-end, rôles utilisateurs et permissions.
- Perspectives futures : Déploiement sur instance cloud, interface multilingue, dashboard d'administration.