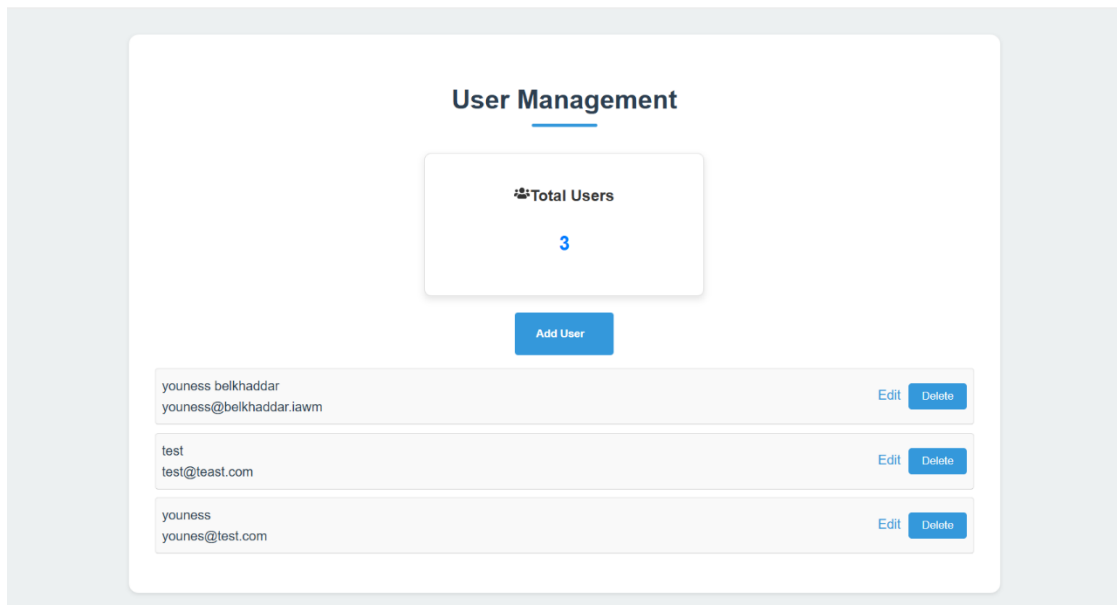


# Rapport Technique - Application de Gestion d'Utilisateurs



## Table des matières

1. [Présentation générale du projet](#)
2. [Architecture technique](#)
3. [Mise en place du backend](#)
4. [Mise en place du frontend](#)
5. [Base de données](#)
6. [Dockerisation](#)
7. [Pipeline CI/CD avec GitHub Actions](#)
8. [Tests et qualité du code](#)
9. [Difficultés rencontrées et solutions](#)
10. [Conclusion et axes d'amélioration](#)

## Présentation générale du projet

L'application de gestion d'utilisateurs est une solution web moderne permettant de gérer efficacement les utilisateurs d'une organisation. Elle offre une interface utilisateur intuitive et des fonctionnalités robustes pour la gestion des comptes utilisateurs.

### Fonctionnalités principales

- Création et gestion des utilisateurs
- Interface utilisateur responsive et moderne
- API RESTful sécurisée
- Authentification des utilisateurs
- Tests automatisés
- Déploiement continu avec Docker

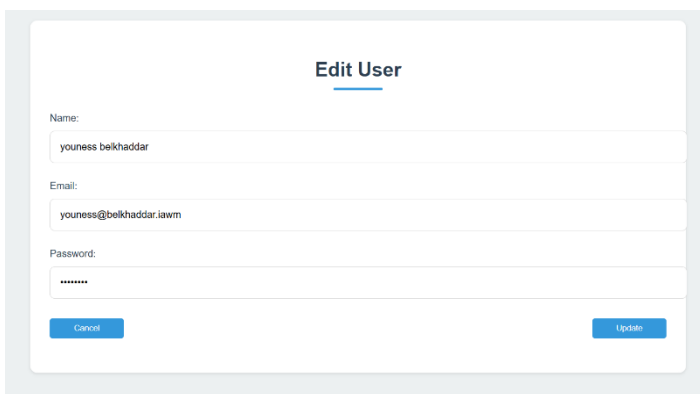


Figure 2 Add and Edit user

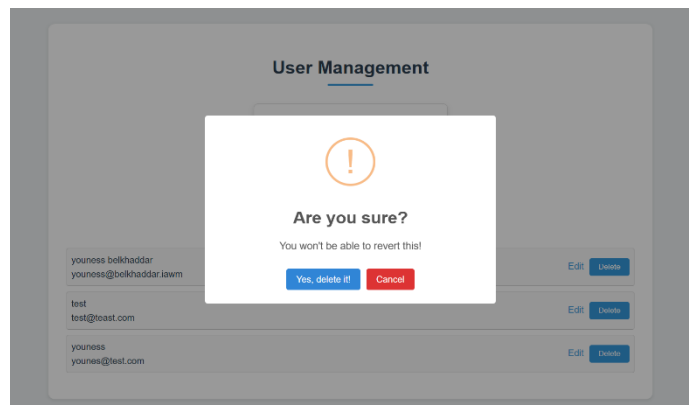


Figure 1 Delete user

## Architecture technique

L'application suit une architecture moderne basée sur le stack MERN (MongoDB, Express.js, React, Node.js) avec une séparation claire entre le frontend et le backend.

### Stack technologique

- **Frontend:** React.js avec Material-UI
- **Backend:** Node.js avec Express.js
- **Base de données:** MySQL
- **Tests:** Jest, Chai, Supertest

**Conteneurisation:** Docker

**CI/CD:** GitHub Actions

## Mise en place du backend

### Structure du backend

```
backend/  
├── src/  
│   ├── config/  
│   ├── controllers/  
│   ├── middleware/  
│   ├── models/  
│   ├── routes/  
│   └── app.js  
├── test/  
└── package.json
```

### Points clés de l'implémentation

1. Configuration de l'environnement avec dotenv
2. Mise en place des middlewares de sécurité
3. Implémentation des routes RESTful
4. Gestion des erreurs centralisée
5. Tests automatisés

## Mise en place du frontend

### Structure du frontend

```
frontend/  
├── src/  
│   ├── components/  
│   ├── services/  
│   ├── styles/  
│   └── App.js  
└── package.json
```

### Caractéristiques principales

1. Interface utilisateur moderne avec Material-UI
2. Gestion d'état avec React Hooks
3. Appels API avec Axios
4. Styles modulaires avec CSS
5. Validation des formulaires

# Base de données

## Schéma de la base de données

```
CREATE TABLE users (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  name VARCHAR(255) NOT NULL,  
  email VARCHAR(255) NOT NULL UNIQUE,  
  password VARCHAR(255) NOT NULL,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

## Configuration

- Base de données: MySQL
- Port: 3306
- Nom de la base: users\_management
- Utilisation de migrations pour la gestion du schéma

# Dockerisation

## Structure Docker

```
.  
├─ docker-compose.yml  
├─ backend/  
│   └─ Dockerfile  
└─ frontend/  
    └─ Dockerfile
```

## Configuration

### Docker

#### 1. Backend Dockerfile

- Image de base: Node.js
- Installation des dépendances
- Configuration de l'environnement
- Exposition du port 5000

#### 2. Frontend Dockerfile

- Image de base: Node.js
- Build de l'application
- Serveur nginx pour la production
- Exposition du port 80

### 3. Docker Compose

- Orchestration des services
- Configuration des réseaux
- Gestion des volumes

Variables d'environnement

## Pipeline CI/CD avec GitHub Actions

### Étapes du pipeline

#### 1. Tests Backend

- Installation des dépendances
  - Exécution des tests unitaires
- Vérification de la couverture

#### 2. Tests Frontend

- Installation des dépendances
  - Build de l'application
- Tests des composants

#### 3. Build et Push Docker

- Construction des images
  - Push vers Docker Hub
- Tagging des versions

#### 4. Déploiement

- Déploiement automatique
  - Vérification de la santé
- Rollback en cas d'échec

## Tests et qualité du code

### Stratégie de test

#### 1. Tests unitaires

- Contrôleurs
  - Services
- Composants React

#### 2. Tests d'intégration

- API endpoints

- Flux utilisateur
- Base de données

### 3. Tests de qualité

- Linting (ESLint)
- Formatage (Prettier)
- Couverture de code

```

PS C:\Users\YOUNE\Desktop\Users-Management> cd backend
PS C:\Users\YOUNE\Desktop\Users-Management\backend> npm test

> users-management-backend@1.0.0 test
> mocha --timeout 10000 test/**/*.test.js

User API Tests
Database connection successful
POST /api/users
2025-04-08T15:53:30.164Z - POST /api/users
Request body: {
  name: 'Test User',
  email: 'test@example.com',
  password: 'password123'
}
✓ should create a new user
2025-04-08T15:53:30.199Z - POST /api/users
Request body: {
  name: 'Test User',
  email: 'test@example.com',
  password: 'password123'
}
2025-04-08T15:53:30.207Z - POST /api/users
Request body: {
  name: 'Test User',
  email: 'test@example.com',
  password: 'password123'
}
✓ should not create a user with an existing email
GET /api/users
2025-04-08T15:53:30.227Z - POST /api/users
Request body: { name: 'User 1', email: 'user1@example.com', password: 'pass1' }
2025-04-08T15:53:30.236Z - POST /api/users
Request body: { name: 'User 2', email: 'user2@example.com', password: 'pass2' }
2025-04-08T15:53:30.245Z - GET /api/users
Request params: {}
✓ should get all users
GET /api/users/:id

```

```

name: 'Updated User',
email: 'updated@example.com',
created_at: '2025-04-08T14:53:30.000Z'
}
✓ should update a user
2025-04-08T15:53:30.346Z - POST /api/users
Request body: { name: 'User 1', email: 'user1@example.com', password: 'pass1' }
2025-04-08T15:53:30.355Z - POST /api/users
Request body: { name: 'User 2', email: 'user2@example.com', password: 'pass2' }
2025-04-08T15:53:30.362Z - PUT /api/users/1
Request body: { name: 'User 1', email: 'user2@example.com', password: 'pass1' }
✓ should not update a user with an existing email
DELETE /api/users/:id
2025-04-08T15:53:30.378Z - POST /api/users
Request body: {
  name: 'Test User',
  email: 'test@example.com',
  password: 'password123'
}
}
}
}
2025-04-08T15:53:30.386Z - DELETE /api/users/1
2025-04-08T15:53:30.386Z - DELETE /api/users/1
2025-04-08T15:53:30.386Z - DELETE /api/users/1
rs/1
rs/1
rs/1
rs/1
Request params: {}
rs/1
Request params: {}
2025-04-08T15:53:30.394Z - GET /api/users/1
Request params: {}
✓ should delete a user

8 passing (292ms)

```

## Difficultés rencontrées et solutions

### 1. Gestion des connexions à la base de données

**Problème:** Fuites de mémoire dans les tests **Solution:**  
Implémentation d'un pool de connexions et nettoyage approprié

### 2. Pipeline CI/CD

**Problème:** Tests qui ne se terminent pas **Solution:** Ajout de `process.exit(0)` après les tests

### 3. Dockerisation

**Problème:** Communication entre conteneurs **Solution:**  
Configuration correcte des réseaux Docker

# Conclusion et axes d'amélioration

## Points forts

- Architecture moderne et évolutive
- Tests automatisés robustes
- Pipeline CI/CD efficace
- Documentation claire

## Axes d'amélioration

### 1. Performance

- Mise en cache
- Optimisation des requêtes
- Compression des assets

### 2. Sécurité

- Authentification JWT
- Rate limiting
- Validation des données

### 3. Fonctionnalités

- Gestion des rôles
- Journalisation avancée
- Interface d'administration

### 4. Monitoring

- Métriques de performance
- Alertes
- Logs centralisés