Projet de Gestion des Utilisateurs

PROJET CONTROLE

Assia OMARI

1. Présentation générale du projet

Dans le cadre de ce projet, j'ai développé une application web de **gestion d'utilisateurs**. Il s'agit d'un système CRUD (Create, Read, Update, Delete) complet, permettant d'enregistrer, consulter, modifier et supprimer des utilisateurs via une interface web.
Ce projet m'a permis de mettre en pratique mes compétences en **développement fullstack** avec les technologies suivantes :

- **Frontend :** React.js (Vite)
- **Backend :** Node.js avec Express
- **Base de données :** PostgreSQL
- **Déploiement local :** Visual Studio Code + Postman pour tester les endpoints
- **Outils :** Git, GitHub, .env, Docker (prévu), CI/CD (prévu avec GitHub Actions)

2. Mise en place du backend

2.1 Initialisation et configuration d'Express.js

J'ai commencé par créer le serveur backend avec les étapes suivantes :

- Initialisation du projet : `npm init`
- Installation des dépendances :
  - `express` pour le serveur web
  - `dotenv` pour gérer les variables d'environnement (configuration PostgreSQL)
  - `pg` pour communiquer avec PostgreSQL
  - `cors` pour permettre l'échange entre mon frontend (port 5173) et backend (port 5174)
- Configuration de `express.json()` pour le parsing des requêtes JSON

Le fichier principal `server.js` inclut toute la configuration du serveur ainsi que la connexion à PostgreSQL.

2.2 Connexion à PostgreSQL

J'ai utilisé la bibliothèque `pg` pour établir une connexion avec une base de données PostgreSQL locale.
La base de données est nommée `users`, avec une table `users` que j'ai créée via une requête SQL exécutée lors du lancement du serveur si elle n'existe pas :

```
sql
CopyEdit
CREATE TABLE IF NOT EXISTS users (
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  email TEXT NOT NULL UNIQUE,
  age INTEGER NOT NULL,
  role TEXT NOT NULL
);
```

Les identifiants de connexion sont stockés dans un fichier `.env` pour plus de sécurité.

2.3 Création des routes REST

J'ai implémenté les routes suivantes :

- `GET /` : Récupération de tous les utilisateurs
- `POST /users` : Ajout d'un nouvel utilisateur
- `PUT /users/:id` : Mise à jour d'un utilisateur
- `DELETE /users/:id` : Suppression d'un utilisateur

Chaque route est testée à l'aide de Postman, avec une gestion claire des erreurs (utilisateur non trouvé, email déjà existant, champs manquants…).

3. Mise en place du frontend avec React

3.1 Initialisation du projet

J'ai créé le frontend avec Vite :

```
bash
CopyEdit
npm create vite@latest gestion-utilisateurs --template react
```

Puis installé les dépendances nécessaires :

- `axios` : pour envoyer les requêtes HTTP à l'API
- `react-router-dom` : pour la navigation entre les composants
- `bootstrap` : pour styliser rapidement les composants

3.2 Structure de l'application

Le projet est structuré comme suit :

- `App.jsx` : Composant principal avec les routes
- `components/` :
  - `UserList.jsx` : Liste des utilisateurs
  - `UserForm.jsx` : Formulaire d'ajout/modification
- `services/api.js` : Centralisation des appels API via axios

3.3 Fonctionnalités frontend

- Affichage dynamique de tous les utilisateurs depuis le backend
- Formulaire réactif pour créer ou modifier un utilisateur
- Suppression avec confirmation
- Navigation fluide entre les composants

Le style est volontairement simple pour se concentrer sur la fonctionnalité.

4. Tests, Git et intégration continue

4.1 Tests manuels

Chaque route backend a été testée avec Postman.
Le frontend a été vérifié sur Chrome et Edge pour s'assurer de la cohérence des opérations CRUD.

4.2 Git & GitHub

Le projet est versionné avec Git. J'ai créé un dépôt GitHub dans lequel j'ai poussé le code. Les commits sont clairs et réguliers.

Un fichier `.gitignore` exclut `node_modules/`, `.env` et autres fichiers sensibles.

4.3 CI/CD

Le projet contenait un fichier `ci.yml` dans `.github/workflows/` qui permettrait l'automatisation de tests ou de déploiements. Comme je ne l'ai pas encore utilisé, je l'ai supprimé pour garder le projet propre.

5. Conclusion

Ce projet m’a permis de consolider mes compétences en :

- Développement backend avec Express.js
- Manipulation d’une base de données PostgreSQL
- Création d’un frontend interactif avec React
- Communication frontend/backend (via API REST)
- Gestion de version et bonnes pratiques de développement

Je suis satisfaite du résultat final, car j’ai réussi à concevoir une application complète, fonctionnelle, et bien structurée. Ce travail m’a aussi permis de mieux comprendre l’importance de l’architecture logicielle et des outils modernes de développement.