Développement d'une application web REST monolithique

Objectif

L'objectif de ce projet est de concevoir, développer et déployer une application web REST monolithique basée sur un sujet d'actualité choisi par l'étudiant. L'application devra démontrer une maîtrise des technologies modernes de développement web, incluant backend, frontend, gestion de base de données, authentification sécurisée, et mise en production automatisée via CI/CD.

Contexte

L'étudiant choisira un sujet d'actualité pertinent (par exemple : gestion d'événements climatiques, suivi de campagnes caritatives, plateforme de sensibilisation à une cause sociale, etc.). L'application devra répondre à un besoin concret lié à ce sujet, en proposant une solution fonctionnelle, sécurisée et bien conçue.

Exigences techniques

L'application devra respecter les spécifications suivantes :

1. Backend

- **Framework** : FastAPI (Python) pour la création d'une API REST monolithique.
- **Base de données** : MySQL pour le stockage des données.
- **Authentification** : Implémentation de JSON Web Tokens (JWT) pour sécuriser les accès (inscription, connexion, gestion des rôles utilisateur).
- **Fonctionnalités minimales** :
    - CRUD (Create, Read, Update, Delete) pour les entités principales du sujet choisi.
    - Gestion des utilisateurs (inscription, connexion, mise à jour du profil).
    - Au moins une fonctionnalité spécifique liée au sujet (exemple : pour une plateforme de dons, un système de suivi des contributions).

2. Frontend

- **Framework** : React.js pour une interface utilisateur dynamique et réactive.
- **Fonctionnalités minimales** :
    - Interface utilisateur intuitive pour interagir avec l'API (pages pour inscription/connexion, gestion des données, affichage des informations).
    - Gestion des erreurs et validation des formulaires.
    - Affichage responsive adapté aux différents appareils (mobile, tablette, desktop).

3. Conception

- **Diagrammes UML** :
    - **Diagramme de classes** : Représentation des entités principales, leurs attributs, méthodes et relations (héritage, association, etc.).
    - **Diagramme de cas d'utilisation** : Identification des acteurs (exemple : utilisateur, administrateur) et des interactions avec le système (exemple : consulter des données, modifier un profil).
- Les diagrammes devront être inclus dans la documentation du projet.

4. Déploiement et CI/CD

- **Conteneurisation** : Utilisation de Docker pour encapsuler l'application (backend, frontend, base de données).
- **CI/CD** : Configuration d'un pipeline avec GitHub Actions pour :
    - Tester le code (tests unitaires pour le backend, tests d'intégration si possible).
    - Construire les images Docker.
    - Déployer l'application sur une plateforme de votre choix (exemple : serveur local, Heroku, ou autre service compatible).
- **Gestion du code source** : Utilisation de GitHub pour le versionnement du code.

Livrables

1. **Code source** :
    - Projet bien structuré sur un dépôt GitHub public ou privé.
    - Backend (FastAPI), frontend (React.js), scripts Docker, et configuration CI/CD.
2. **Documentation** :
    - README détaillant l'installation, l'exécution, et les endpoints de l'API.
    - Documentation des choix techniques (pourquoi FastAPI, MySQL, etc.).
    - Inclusion des diagrammes UML (classes et cas d'utilisation).
3. **Rapport** :
    - Explication du sujet choisi et du problème résolu.
    - Description des fonctionnalités implémentées.
    - Analyse des défis rencontrés et solutions apportées.
4. **Démonstration** :
    - Une vidéo ou une présentation montrant l'application en fonctionnement.

Note : le code et le rapport doivent être déployer dans Github.