Contents

Documentation Technique - EducAI

Table des matières

1. Vue d'ensemble

EducAI est une application web d'assistance pédagogique basée sur l'intelligence artificielle, conçue pour la génération et la correction d'exercices mathématiques. L'application utilise une architecture moderne avec des composants backend et frontend séparés.

1.1 Objectifs techniques

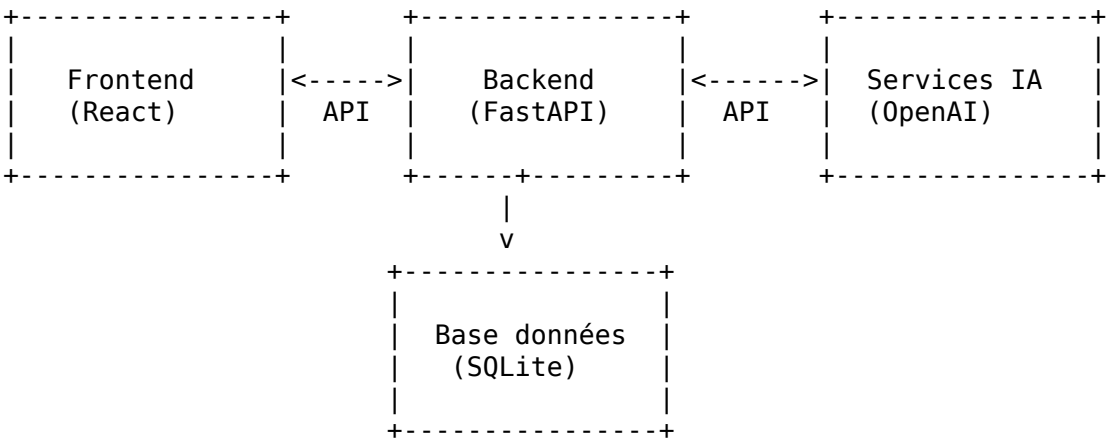
- Fournir une interface utilisateur intuitive pour les professeurs
- Implémenter un système robuste de génération d'exercices via IA
- Permettre la correction automatique des travaux d'élèves
- Gérer efficacement les données des utilisateurs, exercices et résultats

1.2 Technologies principales

- **Backend** : Python (FastAPI)
- **Frontend** : React (JavaScript)
- **Base de données** : SQLite
- **IA** : OpenAI API (intégration GPT)
- **Conteneurisation** : Docker

2. Architecture du système

Le système suit une architecture client-serveur avec séparation claire des responsabilités :



2.1 Flux de données

1. L'utilisateur interagit avec l'interface React
2. Les requêtes sont envoyées au backend FastAPI
3. Le backend communique avec l'API OpenAI si nécessaire
4. Les données sont persistées dans SQLite
5. Les résultats sont renvoyés au frontend

3. Backend

Le backend est construit avec FastAPI, un framework Python moderne pour les API REST.

3.1 Structure des dossiers

```
backend/
├── server.py            # Point d'entrée principal API
├── database.py          # Configuration connexion BD
├── models.py            # Modèles SQLAlchemy
├── utils.py             # Fonctions utilitaires
├── llm_agent.py         # Intégration IA avec OpenAI
├── database model definition/ # Définitions MCD/MLD
├── submissions/         # Stockage soumissions PDF
├── uploads/             # Stockage fichiers uploads
├── requirements.txt     # Dépendances Python
└── Dockerfile           # Configuration Docker
```

3.2 Composants principaux

- **server.py** : Routes API, middleware, authentication
- **database.py** : Configuration SQLAlchemy, session management
- **models.py** : Définition des modèles de données (ORM)
- **llm_agent.py** : Intégration avec l'API OpenAI pour génération de contenu
- **utils.py** : Fonctions utilitaires (traitement PDF, transformations données)

3.3 Dépendances clés

- fastapi : Framework API
- sqlalchemy : ORM pour base de données
- pydantic : Validation de données
- python-multipart : Traitement upload fichiers
- passlib + jwt : Authentification et sécurité
- PyMuPDF (fitz) : Extraction texte PDF
- openai : Intégration API IA

4. Frontend

Le frontend est développé avec React et utilise Chakra UI pour l'interface utilisateur.

4.1 Structure des dossiers

```
eduiafront/
├── src/
│   ├── assets/           # Images, icônes
│   │   ├── Auth/         # Composants authentification
│   │   ├── Chat/         # Interface de chat
│   │   ├── Layout/       # Mise en page, Dashboard
│   │   └── common/       # Composants partagés
│   ├── pages/            # Pages principales
│   ├── services/         # Services API
│   ├── App.jsx           # Composant racine
│   └── main.jsx          # Point d'entrée
├── public/               # Fichiers statiques
├── package.json          # Dépendances
└── Dockerfile            # Configuration Docker
```

4.2 Composants principaux

- **Chat** : Interaction conversationnelle avec l'IA
- **DashboardLayout** : Interface tableau de bord professeur
- **Auth** : Gestion authentification (login/register)
- **CurriculumUpload** : Upload fichier programme

4.3 Dépendances clés

- react : Bibliothèque UI
- chakra-ui : Composants UI
- react-router-dom : Routage
- axios : Client HTTP
- react-icons : Icônes

5. Modèles de données

Les modèles suivent une structure relationnelle, implémentée avec SQLAlchemy.

5.1 Diagramme simplifié des entités

```
Professeur (1) ---< Exercice (1) ---< QCM (1) ---< QCMReponse (n)
                                    |
                                    v
Eleve (1) ---< Soumission (n) ---< Resultat (n)
```

5.2 Tables principales

- **Professeur** : Gestion des comptes enseignants
- **Eleve** : Informations sur les élèves
- **Programme** : Curriculum et programme scolaire
- **Exercice** : Exercices créés par les professeurs
- **QCM** : Questions à choix multiples
- **QCMReponse** : Réponses possibles aux QCM
- **Soumission** : Réponses soumises par les élèves
- **Resultat** : Résultats des évaluations

6. API

L'API est organisée par ressources et suit les principes REST.

6.1 Points de terminaison principaux

Endpoint	Méthode	Description	Authentification
/register/	POST	Inscription professeur	Non
/login/	POST	Authentification	Non
/me/	GET	Profil utilisateur	Oui
/chat/	POST	Interaction chat IA	Oui
/correct-exam/	POST	Correction copie	Oui
/save-result/	POST	Sauvegarde résultat	Oui
/upload-curriculum/	POST	Upload programme	Oui
/students	GET	Liste des élèves	Oui

Endpoint	Méthode	Description	Authentification
/exercises	GET	Liste des exercices	Oui
/metrics	GET	Métriques et stats	Oui
/exams	GET	Liste des examens	Oui
/recommendations/student/{id}	GET	Recommandations	Oui

6.2 Authentification

- Basée sur JSON Web Tokens (JWT)
- Token valide pendant 60 minutes
- Schéma : Bearer token dans header Authorization

7. Intégration IA

L'intelligence artificielle est au cœur du système et s'appuie sur OpenAI.

7.1 Fonctionnalités IA

- **Génération d'exercices** : Création d'exercices et QCM
- **Correction automatique** : Analyse des copies d'élèves
- **Recommandations** : Conseils personnalisés pour élèves
- **Extraction de texte** : Analyse du contenu PDF

7.2 Composants IA

- **invoke_generate_qcm_agent** : Génération d'exercices
- **invoke_analyze_student_copy_agent** : Analyse de copies
- **invoke_llm_recommendation_agent** : Génération de recommandations

7.3 Intégration technique

- Utilisation de l'API OpenAI avec modèle GPT
- Prompt engineering pour contextualiser les demandes
- Structuration des résultats en JSON pour traitement

8. Déploiement

Le déploiement est facilité par Docker et Docker Compose.

8.1 Conteneurs

- **api** : Backend FastAPI
- **frontend** : Interface React

8.2 Configuration docker-compose.yml

```yaml
services:
  api:
    build: ./backend
    ports:
      - "8000:8000"
```

```
    volumes:
      - ./backend:/app
    env_file:
      - .env

  frontend:
    build: ./eduiafront
    ports:
      - "3000:80"
    depends_on:
      - api
```

8.3 Variables d'environnement

- OPENAI_API_KEY : Clé API OpenAI (requise)

9. Sécurité

9.1 Authentification et autorisation

- Mots de passe hachés avec bcrypt
- JWTs pour session utilisateur
- Vérification des droits d'accès aux ressources

9.2 Protection des données

- Validation des entrées utilisateur avec Pydantic
- Échappement des données côté frontend
- Transactions sécurisées avec SQLAlchemy

9.3 Considérations RGPD

- Stockage minimal des données personnelles
- Séparation des données sensibles
- Pas de stockage externe hors application

10. Considérations techniques

10.1 Performance

- SQLite utilisé pour simplicité mais limité en concurrence
- Optimisation des requêtes avec indexation
- Chargement paresseux des données côté frontend

10.2 Limitations connues

- Pas de haute disponibilité (single instance)
- Limites de l'API OpenAI (quotas, latence)
- SQLite en production limité en concurrence
- Tests automatisés incomplets

10.3 Améliorations futures possibles

- Migration vers PostgreSQL pour plus de robustesse
- Implémentation de tests automatisés plus complets
- Mise en cache des réponses IA fréquentes
- Système de stockage fichiers plus robuste (S3/MinIO)
- WebSockets pour communication en temps réel