Smart conseil

Book Management Application – Fullstack (Angular 16 + Flask)

Auteur: Mehdi Ben Fekhta
Date: 03/10/2025

1. Objectif

Mini application CRUD pour gérer une collection de livres.

Stack technique : Front-end Angular 16, Back-end Flask RESTful, Base de données PostgreSQL (Nean db - online), Déploiement via Docker.

2. Fonctionnalités

Front-end (Angular 16):

- Liste des livres (titre, auteur, année)
- Ajout d'un livre via formulaire réactif avec validation
- Modification et suppression d'un livre
- Communication avec l'API Flask (GET, POST, PUT, DELETE)

Back-end (Flask):

- Routes RESTful exposant les endpoints CRUD
- ORM: SQLAlchemy
- Base de données: PostgreSQL (Nean db - online)

3. Modèle Conceptuel de Données (MCD)

Entité principale : Livre

Attributs: id, titre, auteur, annee, book_image (optionnel)

Remarque: un PDF du MCD (MCD_Books.pdf) peut être inclus dans le repo pour visualisation.

4. Structure du projet

Arborescence recommandée:

```
project/
├─ backend/
│  ├─ app.py
│  ├─ requirements.txt
│  └─ Dockerfile
├─ frontend/
│  ├─ angular.json
│  ├─ package.json
│  ├─ src/
│  │   ├─ app/
│  │   ├─ assets/
│  │   └─ index.html
│  └─ Dockerfile
├─ docker-compose.yml
├─ README.md
└─ MCD_Books.pdf
```

5. Installation et exécution

Avec Docker (recommandé):

1. git clone <votre-repo-url>
2. cd project
3. docker-compose up --build
4. Front-end: http://localhost:4200
5. Back-end: http://localhost:5000

Sans Docker - Backend (Flask):

6. cd backend
7. python -m venv venv
8. source venv/bin/activate   # Linux/macOS

```
9. venv\Scripts\activate     # Windows
10.  pip install -r requirements.txt
11.  python app.py  # API available at http://localhost:5000
```

Sans Docker - Frontend (Angular):

```
12.  cd frontend
13.  npm install
14.  ng serve  # Front-end at http://localhost:4200
```

6. API Endpoints

Méthode	Route	Description
GET	/books	Liste des livres
POST	/books	Ajouter un livre
PUT	/books/<id>	Modifier un livre
DELETE	/books/<id>	Supprimer un livre

7. Critères d'évaluation

- CRUD opérationnel et testable
- Formulaire Angular réactif avec validation
- MCD correctement implémenté
- Code propre et structuré (front + back)
- Déploiement Docker fonctionnel
- Documentation claire

8. Docker (exemples)

Backend Dockerfile (Flask):

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

Frontend Dockerfile (Angular):

```
FROM node:20 as build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build --prod

FROM nginx:alpine
COPY --from=build /app/dist/<your-angular-app-name>
/usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

docker-compose.yml:

```
version: '3.9'
services:
  backend:
    build: ./backend
    ports:
      - "5000:5000"
    volumes:
      - ./backend:/app
  frontend:
    build: ./frontend
    ports:
      - "4200:80"
    depends_on:
      - backend
```

9. Notes finales

Remarques et bonnes pratiques :

- - Prévoir validations côté backend et frontend (WTForms/Marshmallow ou Flask-RESTful serializers + Angular validators).
- - Sécuriser l'API (CORS).
- - Prévoir tests unitaires / e2e pour le backend et le frontend.