

IV. Création d'une api REST avec FastAPI

1. Introduction à FastAPI

- Framework moderne pour construire des API web en Python
- Basé sur :
 - **Starlette** (serveur web async hautes performances)
 - **Pydantic** (validation et sérialisation des données)
- Avantages :
 - Documentation Swagger/OpenAPI automatique
 - Support natif de l'asynchrone
 - Développement rapide et lisible

2. Définir des endpoints

- Chaque **route** (ou endpoint) représente une action de l'API
- On associe une **fonction Python** à une **méthode HTTP** (GET, POST...)
- Les routes peuvent avoir :
 - des **paramètres d'URL** (`/users/{id}`)
 - des **paramètres de requête**
 - un **corps JSON** à valider

3. Liaison avec SQLModel

- Récupérer ou insérer des objets SQLModel directement dans l'API
- Utilisation de la session pour requêter la base
- Sérialisation automatique des objets SQLModel en JSON

4. Dépendances et injection

- Mécanisme de **Depends()** pour injecter automatiquement des dépendances (ex : la session de base de données)
- Centralisation des connexions, autorisations, logique répétitive
- Évite le couplage fort entre les fonctions

5. Validation et documentation

- Pydantic valide automatiquement les entrées (types, champs obligatoires)
- Documentation automatique générée :
 - `/docs` (Swagger UI)
 - `/redoc`

- Schémas clairs des modèles d'entrée/sortie

06. Gestion des erreurs

- Utilisation de **HTTPException** pour renvoyer des erreurs HTTP appropriées
- Codes d'erreur standard (404, 422, etc.)
- Messages d'erreur clairs pour les développeurs
- Gestion des exceptions globales avec erreurs personnalisées
- exception_handler pour gérer les erreurs de manière centralisée

07. Organisation du projet

- Séparer les **routes, services, modèles, schemas**
- Exemple d'arborescence :

```
app/
├── main.py
├── models/
├── routes/
├── db/
├── schemas/
└── services/
```

- recommendations