Code de l'API : fichier main.py

Nous allons construire pas à pas notre API avec FastAPI en ajoutant progressivement les Endpoints (points de terminaison).

Endoint de vérification que l'API MovieLens fonctionne

```
from fastapi import FastAPI, Depends, HTTPException, Query, Path
from sqlalchemy.orm import Session
from typing import List, Optional

from database import SessionLocal
import models
import query_helpers as helpers
import schemas  # Ajouté pour utiliser les schémas Pydantic

# Initialisation de l'application FastAPI
app = FastAPI(
    title="MovieLens API",
    description="API pour interroger la base de données MovieLens",
    version="0.1"
)

# Dépendance pour récupérer une session DB
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

# Endpoint de health check
@app.get(
    "/",
    summary="Vérifie si l'API MovieLens fonctionne",
    description="""
    Ce point d'entrée permet de vérifier si l'API MovieLens est
opérationnelle.
    """,
    response_description="Un message de confirmation si l'API fonctionne
correctement.",
    operation_id="health_check_movies_api",
    tags=["monitoring"],
)
async def root():
    return {"message": "API MovieLens opérationnelle"}
```

Exécutons FastAPI en mode développement avec rechargement automatique à chaque modification du fichier *main.py* :

```
(.venv) vant@MOOVE15:~/Documents/business/movielens-project$ cd api
(.venv) vant@MOOVE15:~/Documents/business/movielens-project/api$ uvicorn
main:app --reload
```

Accédons ensuite à l'API : http://127.0.0.1:8000/

L'API s'ouvre dans le navigateur par défaut de notre machine et on voit bien apparaître : {"message":"API MovieLens opérationnelle"}

On peut aussi accéder à la doc interactive Swagger sur http://127.0.0.1:8000/docs

Rajoutons un autre point de terminaison.

Endpoint pour obtenir un film par son identifiant

```
# Endpoint pour obtenir un film par son ID
@app.get(
    "/movies/{movie_id}",
    summary="Obtenir un film par son ID",
    description="Retourne les informations d'un film en utilisant son
`movieId`.",
    response_description="Détails du film",
    response_model=schemas.MovieDetailed,
    tags=["films"],
)
def read_movie(movie_id: int = Path(..., description="L'identifiant unique
du film"), db: Session = Depends(get_db)):
    movie = helpers.get_movie(db, movie_id)
    if movie is None:
        raise HTTPException(status_code=404, detail=f"Film avec l'ID
{movie_id} non trouvé")
    return movie
```

Si besoin actualiser l'API au niveau du navigateur.

Au cours du développement de l'API, on peut tester un point de terminaison de deux manières :

Méthode 1 : Utiliser Swagger UI (recommandée)

1. **Lance ton app** (si ce n'est pas déjà fait) :

   ```
   uvicorn main:app --reload
   ```

2. Va dans ton navigateur à l'adresse :

   ```
   http://127.0.0.1:8000/docs
   ```

3. Tu verras une interface interactive avec **tous tes endpoints**. Clique sur :
    - **GET** `/movies/{movie_id}`
    - Clique sur le bouton **"Try it out"**
    - Entre un `movie_id` (par exemple `1`)
    - Clique sur **"Execute"**
4. Tu verras la requête envoyée, la réponse JSON, et le code de retour HTTP (200, 404, etc.)

Méthode 2 : Entrer directement l'URL dans le navigateur

Si tu veux tester sans passer par Swagger, tape simplement dans la barre d'adresse :

```
http://127.0.0.1:8000/movies/1
```

(remplace `1` par n'importe quel `movie_id` qui existe dans ta base)

Remarques

- Si tu reçois une **erreur 404**, c'est que :
    - Le `movie_id` n'existe pas dans ta base
    - Ou bien l'endpoint `/movies/{movie_id}` n'est pas encore bien défini dans `main.py`

Pour tous les autres tests de point de terminaison, on utilisera Swagger.

Rajoutons un autre point de terminaison.

Endpoint pour obtenir une liste des films (avec pagination et filtres facultatifs title, genre, skip, limit)

```
@app.get(
    "/movies",
    summary="Lister les films",
    description="Retourne une liste de films avec pagination et filtres
optionnels par titre ou genre.",
    response_description="Liste de films",
    response_model=List[schemas.MovieSimple],
    tags=["films"],
)
def list_movies(
    skip: int = Query(0, ge=0, description="Nombre de résultats à
ignorer"),
    limit: int = Query(100, le=1000, description="Nombre maximal de
résultats à retourner"),
    title: str = Query(None, description="Filtre par titre"),
    genre: str = Query(None, description="Filtre par genre"),
```

```
    db: Session = Depends(get_db)
):
    movies = helpers.get_movies(db, skip=skip, limit=limit, title=title,
genre=genre)
    return movies
```

Pour tester le endpoint `/movies`, voici des **exemples de requêtes HTTP** que tu peux entrer dans le navigateur ou via Swagger UI :

---

1. Obtenir la liste des 100 premiers films (valeurs par défaut)

URL :

```
http://127.0.0.1:8000/movies
```

Résultat attendu :
Une liste de 100 films (ou moins s'il y en a moins en base).

---

2. Pagination – Obtenir les 20 films suivants après les 100 premiers

URL :

```
http://127.0.0.1:8000/movies?skip=100&limit=20
```

Résultat :
Les films allant de l'index 101 à 120.

---

3. Filtrer par titre (recherche partielle)

URL :

```
http://127.0.0.1:8000/movies?title=story
```

Résultat :
Tous les films dont le titre contient "story" (ex: *Toy Story*, *Love Story*…).

---

4. Filtrer par genre

URL :

```
http://127.0.0.1:8000/movies?genre=Comedy
```

Résultat :
Tous les films dont le genre contient “Comedy”.

---

5. Combiner filtres : titre + genre + pagination

URL :

```
http://127.0.0.1:8000/movies?title=story&genre=Adventure&skip=0&limit=5
```

Résultat :
Les 5 premiers films dont le titre contient “story” **et** le genre contient “Adventure”.

Tu peux aussi faire tout ça dans Swagger UI :
http://127.0.0.1:8000/docs → sélectionne `/movies` → *Try it out*

Rajoutons un autre point de terminaison dans le fichier main.py

Endpoint pour obtenir une évaluation par utilisateur et film

Objectif de l’endpoint : Récupérer une **évaluation (rating)** donnée par un utilisateur (`user_id`) pour un film (`movie_id`).

```
# Endpoint pour obtenir une évaluation par utilisateur et film
@app.get(
    "/ratings/{user_id}/{movie_id}",
    summary="Obtenir une évaluation par utilisateur et film",
    description="Retourne l'évaluation d'un utilisateur pour un film
donné.",
    response_description="Détails de l'évaluation",
    response_model=schemas.RatingSimple,
    tags=["évaluations"],
)
def read_rating(
    user_id: int = Path(..., description="ID de l'utilisateur"),
    movie_id: int = Path(..., description="ID du film"),
    db: Session = Depends(get_db)
):
    rating = helpers.get_rating(db, user_id=user_id, movie_id=movie_id)
    if rating is None:
        raise HTTPException(
            status_code=404,
            detail=f"Aucune évaluation trouvée pour l'utilisateur {user_id}
et le film {movie_id}"
```

```
        )
    return rating
```

Voici des exemples concrets que tu peux tester dans ton navigateur pour le nouvel **endpoint** `/ratings/{user_id}/{movie_id}`.

Format URL :

```
http://localhost:8000/ratings/{user_id}/{movie_id}
```

Cas 1 : L'utilisateur a évalué le film

Exemple URL (à adapter à ta base de données) :

```
http://localhost:8000/ratings/1/1
```

Cela recherche l'évaluation laissée par **l'utilisateur 1** pour **le film 1**.

Réponse JSON attendue :

```
{
  "userId": 1,
  "movieId": 1,
  "rating": 4.0,
  "timestamp": 964982703
}
```

Cas 2 : L'utilisateur N'A PAS évalué ce film

Exemple URL :

```
http://localhost:8000/ratings/1/999999
```

Réponse attendue :

```
{
  "detail": "Aucune évaluation trouvée pour l'utilisateur 1 et le film
999999"
}
```

Statut HTTP : `404 Not Found`

Cas 3 : L'`user_id` ou le `movie_id` n'existe pas dans la base

Exemple :

```
http://localhost:8000/ratings/999999/999999
```

Même résultat que le cas 2 : **404 Not Found**

Cas 4 : Valeurs invalides (ex : texte au lieu d'un entier)

Exemple :

```
http://localhost:8000/ratings/abc/def
```

FastAPI va répondre automatiquement avec un message d'erreur de validation :

```
{
  "detail": [
    {
      "loc": ["path", "user_id"],
      "msg": "value is not a valid integer",
      "type": "type_error.integer"
    },
    {
      "loc": ["path", "movie_id"],
      "msg": "value is not a valid integer",
      "type": "type_error.integer"
    }
  ]
}
```

Statut HTTP : `422 Unprocessable Entity`

Astuce : Utiliser l'interface Swagger UI

Tu peux aussi **tester facilement cet endpoint dans ton navigateur à l'adresse :**

```
http://localhost:8000/docs
```

Clique sur `GET /ratings/{user_id}/{movie_id}` → **Try it out** → entre des valeurs → **Execute**

Rajoutons un autre point de terminaison dans le fichier main.py

Endpoint pour obtenir une liste d'évaluations avec pagination et filtres optionnels (film, utilisateur, note min)

```
GET /ratings
```

Ce endpoint permet de lister les évaluations avec les **filtres optionnels** suivants :

- `skip` (int) : nombre d'éléments à ignorer (pagination)
- `limit` (int) : nombre maximum d'évaluations à retourner
- `movie_id` (int) : filtre par identifiant de film
- `user_id` (int) : filtre par identifiant d'utilisateur
- `min_rating` (float) : filtre pour ne retourner que les notes ≥ min_rating

URL de base (local)

```
http://localhost:8000/ratings
```

```
# Endpoint pour obtenir une liste d'évaluations avec filtres
@app.get(
    "/ratings",
    summary="Lister les évaluations",
    description="Retourne une liste d'évaluations avec pagination et
filtres optionnels (film, utilisateur, note min).",
    response_description="Liste des évaluations",
    response_model=List[schemas.RatingSimple],
    tags=["évaluations"],
)
def list_ratings(
    skip: int = Query(0, ge=0, description="Nombre de résultats à
ignorer"),
    limit: int = Query(100, le=1000, description="Nombre maximal de
résultats à retourner"),
    movie_id: Optional[int] = Query(None, description="Filtrer par ID de
film"),
    user_id: Optional[int] = Query(None, description="Filtrer par ID
d'utilisateur"),
    min_rating: Optional[float] = Query(None, ge=0.0, le=5.0,
description="Filtrer les notes supérieures ou égales à cette valeur"),
    db: Session = Depends(get_db)
):
    ratings = helpers.get_ratings(db, skip=skip, limit=limit,
```

```
    movie_id=movie_id, user_id=user_id, min_rating=min_rating)
    return ratings
```

Voici **tous les cas possibles** pour tester le endpoint `GET /ratings` directement dans le navigateur ou via l'interface Swagger de FastAPI.

Cas 1 : Liste des 100 premières évaluations

```
http://localhost:8000/ratings
```

Comportement attendu : retourne 100 évaluations (par défaut).

Cas 2 : Pagination — 10 évaluations à partir de la 50e

```
http://localhost:8000/ratings?skip=50&limit=10
```

Retourne les évaluations 51 à 60.

Cas 3 : Filtrer par `movie_id` (ex : film 1)

```
http://localhost:8000/ratings?movie_id=1
```

Retourne les évaluations pour le **film 1** uniquement.

Cas 4 : Filtrer par `user_id` (ex : utilisateur 2)

```
http://localhost:8000/ratings?user_id=2
```

Retourne toutes les évaluations faites par l'utilisateur 2.

Cas 5 : Filtrer par `min_rating` (ex : note ≥ 4.5)

```
http://localhost:8000/ratings?min_rating=4.5
```

Retourne toutes les évaluations ayant une note **≥ 4.5**.

Cas 6 : Filtre combiné — utilisateur 1, film 1, note ≥ 3

```
http://localhost:8000/ratings?user_id=1&movie_id=1&min_rating=3
```

Retourne l'évaluation de l'utilisateur 1 pour le film 1 si elle est ≥ 3.

Cas 7 : Aucun résultat (valeurs valides mais sans correspondance)

```
http://localhost:8000/ratings?user_id=123456&movie_id=654321
```

Réponse :

```
[]
```

Statut HTTP : `200 OK` (liste vide)

Cas 8 : Valeurs invalides

```
http://localhost:8000/ratings?user_id=abc
```

FastAPI retourne une erreur de validation :

```
{
  "detail": [
    {
      "type": "int_parsing",
      "loc": [
        "query",
        "user_id"
      ],
      "msg": "Input should be a valid integer, unable to parse string as an
integer",
      "input": "abc"
    }
  ]
}
```

Statut HTTP : `422 Unprocessable Entity`

Rajoutons un autre point de terminaison dans le fichier main.py

Endpoint pour retourner un tag pour un utilisateur et un film donnés, avec le texte du tag

```
#  Endpoint pour retourner un tag pour un utilisateur et un film donnés,
avec le texte du tag
@app.get(
    "/tags/{user_id}/{movie_id}/{tag_text}",
    summary="Obtenir un tag spécifique",
    description="Retourne un tag pour un utilisateur et un film donnés,
avec le texte du tag.",
    response_model=schemas.TagSimple,
    tags=["tags"],
)
def read_tag(
    user_id: int = Path(..., description="ID de l'utilisateur"),
    movie_id: int = Path(..., description="ID du film"),
    tag_text: str = Path(..., description="Contenu exact du tag"),
    db: Session = Depends(get_db)
):
    result = helpers.get_tag(db, user_id=user_id, movie_id=movie_id,
tag_text=tag_text)
    if result is None:
        raise HTTPException(
            status_code=404,
            detail=f"Tag non trouvé pour l'utilisateur {user_id}, le film
{movie_id} et le tag '{tag_text}'"
        )
    return result
```

Voici un exemple de test : http://localhost:8000/tags/2/60756/funny et voici le résultat :


```
{
  "userId": 2,
  "movieId": 60756,
  "tag": "funny",
  "timestamp": 1445714994
}
```


Rajoutons un autre point de terminaison dans le fichier main.py

---

Endpoint pour retourner une liste de tags avec pagination et filtres facultatifs par utilisateur ou film

```
# Endpoint pour retourner une liste de tags avec pagination et filtres
facultatifs par utilisateur ou film
@app.get(
```

```
    "/tags",
    summary="Lister les tags",
    description="Retourne une liste de tags avec pagination et filtres
facultatifs par utilisateur ou film.",
    response_model=List[schemas.TagSimple],
    tags=["tags"],
)
def list_tags(
    skip: int = Query(0, ge=0, description="Nombre de résultats à
ignorer"),
    limit: int = Query(100, le=1000, description="Nombre maximal de
résultats à retourner"),
    movie_id: Optional[int] = Query(None, description="Filtrer par ID de
film"),
    user_id: Optional[int] = Query(None, description="Filtrer par ID
d'utilisateur"),
    db: Session = Depends(get_db)
):
    return helpers.get_tags(db, skip=skip, limit=limit, movie_id=movie_id,
user_id=user_id)
```

Voici des Exemple de requêtes pour tester ce endpoint :

- **Tous les tags (limités à 100 par défaut)**

  ```
  GET /tags
  ```

- **Pagination : page 2 (100 premiers ignorés)**

  ```
  GET /tags?skip=100&limit=100
  ```

- **Filtrage par utilisateur** `user_id=2`

  ```
  GET /tags?user_id=2
  ```

- **Filtrage par film** `movie_id=60756`

  ```
  GET /tags?movie_id=60756
  ```

- **Filtrage combiné**

  ```
  GET /tags?user_id=2&movie_id=60756
  ```

Rajoutons un autre point de terminaison dans le fichier main.py

Endpoint pour retourner les identifiants IMDB et TMDB pour un film donné

```
# Endpoint pour retourner les identifiants IMDB et TMDB pour un film donné
@app.get(
    "/links/{movie_id}",
    summary="Obtenir le lien d'un film",
    description="Retourne les identifiants IMDB et TMDB pour un film
donné.",
    response_model=schemas.LinkSimple,
    tags=["links"],
)
def read_link(
    movie_id: int = Path(..., description="ID du film"),
    db: Session = Depends(get_db)
):
    result = helpers.get_link(db, movie_id=movie_id)
    if result is None:
        raise HTTPException(
            status_code=404,
            detail=f"Aucun lien trouvé pour le film avec l'ID {movie_id}"
        )
    return result
```

Voici un exemple de test de ce endpoint `/links/{movie_id}` dans le **navigateur** :

```
http://localhost:8000/links/1
```

Voici le résultat :

```
{
  "movieId": 1,
  "imdbId": "0114709",
  "tmdbId": 862
}
```

Rajoutons un autre point de terminaison dans le fichier main.py

Endpoint pour retourner une liste paginée des identifiants IMDB et TMDB de tous les films

```
# Endpoint pour retourner une liste paginée des identifiants IMDB et TMDB
de tous les films
```

```
@app.get(
    "/links",
    summary="Lister les liens des films",
    description="Retourne une liste paginée des identifiants IMDB et TMDB
de tous les films.",
    response_model=List[schemas.LinkSimple],
    tags=["links"],
)
def list_links(
    skip: int = Query(0, ge=0, description="Nombre de résultats à
ignorer"),
    limit: int = Query(100, le=1000, description="Nombre maximal de
résultats à retourner"),
    db: Session = Depends(get_db)
):
    return helpers.get_links(db, skip=skip, limit=limit)
```

Exemple d'URL pour tester :

```
http://localhost:8000/links?skip=10&limit=20
```

Rajoutons un autre point de terminaison dans le fichier main.py

Endpoint pour obtenir des statistiques sur la base de données

```
# Endpoint pour obtenir des statistiques sur la base de données
@app.get(
    "/analytics",
    summary="Obtenir les statistiques analytiques",
    description="Retourne le nombre total de films, évaluations, tags et
liens.",
    response_model=schemas.AnalyticsResponse,
    tags=["analytics"]
)
def get_analytics(db: Session = Depends(get_db)):
    movie_count = helpers.get_movie_count(db)
    rating_count = helpers.get_rating_count(db)
    tag_count = helpers.get_tag_count(db)
    link_count = helpers.get_link_count(db)

    return schemas.AnalyticsResponse(
        movie_count=movie_count,
        rating_count=rating_count,
        tag_count=tag_count,
        link_count=link_count
    )
```

Pour tester : http://127.0.0.1:8000/analytics

Résultat : {"movie_count":9742,"rating_count":100836,"tag_count":3683,"link_count":9742}

Documentation de l'API

La **documentation** est une étape essentielle pour rendre votre API facile à comprendre et à utiliser.

FastAPI propose **deux interfaces de documentation intégrées** (ou *built-in*) qu'on peut utiliser sans rien configurer : **Swagger UI** et **ReDoc**. Ces interfaces sont générées automatiquement à partir du code Pythonm (FastAPI) de l'API.

1. Swagger UI (interface par défaut)

Accès : `http://127.0.0.1:8000/docs`

Avec la documentation Swagger, on peut :

- Voir la **liste des endpoints** (`/movies`, `/movies/{movie_id}`, etc.)
- Lire les **descriptions** et les **paramètres** attendus
- Envoyer des requêtes **GET**, **POST**, etc. directement depuis l'interface
- Visualiser les **réponses JSON**

2. ReDoc (documentation plus lisible)

Accès : `http://127.0.0.1:8000/redoc`

- Affichage plus **structuré et professionnel**
- Pratique pour une **lecture approfondie** de l'API
- Très apprécié dans les projets en entreprise

À retenir

Interface	URL	Usage principal
Swagger UI	`/docs`	Tester les endpoints facilement
ReDoc	`/redoc`	Lire la doc proprement

Amélioration de documentation dans le code de main.py

```
api_description = """
Bienvenue dans l'API **MovieLens**

Cette API permet d'interagir avec une base de données inspirée du célèbre
jeu de données [MovieLens](https://grouplens.org/datasets/movielens/).
Elle est idéale pour découvrir comment consommer une API REST avec des
données de films, d'utilisateurs, d'évaluations, de tags et de liens
externes (IMDB, TMDB).
```

```
### Fonctionnalités disponibles :

- Rechercher un film par ID, ou lister tous les films
- Consulter les évaluations (ratings) par utilisateur et/ou film
- Accéder aux tags appliqués par les utilisateurs sur les films
- Obtenir les liens IMDB / TMDB pour un film
- Voir des statistiques globales sur la base

Tous les endpoints supportent la pagination (`skip`, `limit`) et des
filtres optionnels selon les cas.

### Bon à savoir
- Vous pouvez tester tous les endpoints directement via l'interface Swagger
ci-dessous.
- Pour toute erreur (ex : ID inexistant), une réponse claire est retournée
avec le bon code HTTP.
"""

# Initialisation de l'application FastAPI
app = FastAPI(
    title="MovieLens API",
    description=api_description,
    version="0.1"
)


# Endpoint pour obtenir des statistiques sur la base de données
@app.get(
    "/analytics",
    summary="Obtenir des statistiques",
    description="""
    Retourne un résumé analytique de la base de données :

    - Nombre total de films
    - Nombre total d'évaluations
    - Nombre total de tags
    - Nombre de liens vers IMDB/TMDB
    """,
    response_model=schemas.AnalyticsResponse,
    tags=["analytics"]
)
def get_analytics(db: Session = Depends(get_db)):
    movie_count = helpers.get_movie_count(db)
    rating_count = helpers.get_rating_count(db)
    tag_count = helpers.get_tag_count(db)
    link_count = helpers.get_link_count(db)

    return schemas.AnalyticsResponse(
        movie_count=movie_count,
        rating_count=rating_count,
        tag_count=tag_count,
        link_count=link_count
    )
```