Prova. Implementació d'una api amb FastAPI

Important: s'ha de donar suport a la validació de dades. Això vol dir, definir bé els models de dades amb Pydantic, i definir els tipus de dades d'entrada i sortida per a cada endpoint. En cas contrari, la puntuació màxima de cada exercici serà ½ punt.

Pas 1: Settings

1. **Crear un projecte FastAPI:**
    - Instal·leu FastAPI standard o feu servir un venv ja existent.
    - Creeu un fitxer principal (`app.py`).
2. **Definir una ruta d'inici:**
    - A la ruta `/` (root), retorneu un missatge de benvinguda o informació bàsica sobre l'API.
    - Comproveu que tot funciona correctament
3. **Carregar el fitxer json**
    - al fitxer database.py teniu les operacions de lectura i d'escriptura per llegir i escriure un fitxer json.
    - Comproveu que funciona correctament, feu un print de prova. Recordeu que això us retorna un diccionari. Si voleu treballar amb llistes enlloc de diccionaris, ho podeu adaptar fàcilment.

      ```
      DB_PATH = "data/posts.json"
      db = read_db(DB_PATH, "posts")
      ```

    - No cal que feu servir persistència amb write_db(), però comproveu amb Thunder Client que les operacions són correctes.

Pas 2: Data Model

Implementeu el model de dades utilitzant la llibreria Pydantic. Haureu de definir dos models, un d'entrada i un altre de sortida. Feu servir Thunder Client per testejar els endpoints.

1. **Model per a operacions d'entrada (Input Model):**
    - S'utilitzarà per a operacions com la creació de posts.
    - Aquest model no inclou l'`id`, ja que aquest s'haurà de generar automàticament.
2. **Model per a operacions de sortida (Output Model):**
    - S'utilitzarà per a operacions com llegir, actualitzar o eliminar posts.
    - Aquest model heretarà del model d'entrada i afegirà l'atribut `id`.

Pas 3: Implementació de les Operacions Bàsiques [5 punts]

Implementeu els endpoints bàsics per realitzar operacions CRUD sobre el conjunt de dades. .

1. **Llegir totes les posts (Read):**
    - Endpoint GET que retorni la llista completa de posts utilitzant el model `post`.

      ```
      path: "/posts"
      ```

2. **Llegir un post per `id` (Read Item):**
    - Endpoint GET amb un paràmetre de ruta `post_id` que retorni el post corresponent.
    - Gestioneu l'error si no es troba.

      ```
      path: "/posts/{post_id}"
      ```

3. **Crear un post (Create):**
    - Endpoint POST que rep dades segons el model `postInput`.
    - Afegir el post a la llista en memòria assignant-li un `id` únic.

      ```
      path: "/posts"
      ```

4. **Actualitzar un post (Update):**
    - Endpoint PUT que rep un `post_id` i un body amb les noves dades.
    - Actualitzeu el post en la llista si existeix.
    - Gestioneu l'error si no es troba.

      ```
      path: "/posts/{post_id}"
      ```

5. **Eliminar un post (Delete):**
    - Endpoint DELETE que rep un `post_id` i elimina el post corresponent.
    - Gestioneu l'error si el post no existeix.

      ```
      path: "/posts/{post_id}"
      ```

Pas 4: Funcionalitats Addicionals [5 punts]

1. **Filtrar posts per tag (tags):**
    - Permet als usuaris recuperar posts amb un tag determinat. **Recordatori:** podeu comprovar si un element es troba en una llista amb **in**
2. **Filtrar posts per paraula o paraules:**
    - Permet als usuaris llistar els posts que contenen a *title* un determinada expressió.
3. **Cercar els posts d'un usari:**
    - Donat el *id* d'un usuari, recuperar els posts creats per aquest usuari.
4. **Cercar els cinc posts amb més likes:**
    - Aquí s'ha de retornar una llista de 5 diccionaris. Cada diccionari ha de contenir *title*, el número de *likes* i el *id* de l'usuari que va fer el post. Per exemple,

```
{
 "title": "His mother had always taught him",
 "likes": 192,
 "userId": 121
}
```

5. **Cercar els cinc posts amb més views:**
   - Aquí s'ha de retornar una llista de 5 diccionaris. Cada diccionari ha de contenir *title*, el número de *views* i el *id* de l'usuari que va fer el post, i el ratio likes/dislikes. Exemple:

```
{
 "title": "His mother had always taught him",
 "views": 305,
 "userId": 121,
 "ratio": 7.68
}
```

Exemple JSON d'un post:

```
{
 "id": 1,
 "title": "His mother had always taught him",
 "body": "His mother had always taught him not to ever think of
himself as better than others. He'd tried to live by this motto. He
never looked down on those who were less fortunate or who had less
money than him. But the stupidity of the group of people he was
talking to made him change his mind.",
 "tags": [
   "history",
   "american",
   "crime"
 ],
 "reactions": {
   "likes": 192,
   "dislikes": 25
 },
 "views": 305,
 "userId": 121
}
```