API CRUD de Receptes amb FastAPI

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. Això ara no és important, perquè al proper tema veurem persistència de dades i farem servir una base de dades.

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

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 (o Postman) per testejar els endpoints. Feu servir un client en python `client.py` amb crides a la api.

1. **Model per a operacions d'entrada (Input Model):**
    - S'utilitzarà per a operacions com la creació de receptes.
    - Aquest model no inclou l'`id`, ja que aquest es pot generar automàticament.
    - Recordeu que aquest `id` serà gestionat per la base de dades quan incorporem persistència en el proper tema. De moment ho fem nosaltres a mà. Recordeu també, que len() no és una bona estratègia.
2. **Model per a operacions de sortida (Output Model):**
    - S'utilitzarà per a operacions com llegir, actualitzar o eliminar receptes.
    - Aquest model heretarà del model d'entrada i afegirà l'atribut `id`.

Pas 3: Implementació de les Operacions Bàsiques [7 punts]

Implementeu els endpoints bàsics per realitzar operacions CRUD sobre el conjunt de dades. En aquesta fase, la persistència es farà amb el fitxer de json i les operacions `read_db()` i `save_db()`.

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

      ```
      @app.get("/recipes")
      ```

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

      ```
      @app.get("/recipes/{recipe_id}")
      ```

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

      ```
      @app.post("/recipes")
      ```

4. **Actualitzar una recepta (Update):**
    - Endpoint PUT que rep un `recipe_id` i un body amb les noves dades.
    - Actualitzeu la recepta en la llista si existeix.

      ```
      @app.put("/recipes/{recipe_id}")
      ```

5. **Eliminar una recepta (Delete):**
    - Endpoint DELETE que rep un `recipe_id` i elimina la recepta corresponent.
    - Gestioneu l'error si la recepta no existeix.

      ```
      @app.delete("/recipes/{recipe_id}")
      ```

Pas 4: Funcionalitats Addicionals [3 punts]

1. **Filtrar receptes per cuina (cuisine):**
    - Permet als usuaris recuperar receptes d'una cuina específica.
2. **Filtrar receptes per dificultat:**
    - Permet als usuaris llistar receptes segons el nivell de dificultat (Fàcil, Mitjà, Difícil).
3. **Cercar receptes per ingredient:**
    - Proporciona un endpoint per cercar receptes que continguin un ingredient determinat.
4. **Filtrar receptes per temps màxim de preparació:**
    - Permet als usuaris trobar receptes que es preparin en un temps màxim determinat.
5. **Obtenir receptes per tipus de menjar:**
    - Permet als usuaris consultar receptes segons el tipus de menjar (Esmorzar, Dinar, Sopar).
6. **Ordenar receptes per calories per ració:**

- Permet als usuaris ordenar les receptes per calories per ració, de manera ascendent o descendent.

Definiu paths per aquestes noves funcionalitats. Per example:

```
@app.get("/recipes/cuisine/{cuisine_name}")

@app.get("/recipes/difficulty/{difficulty}")
```

Per donar suport a la validació de dades, definiu també els tipus d'entrada i sortida per cada endpoint.

Exemple JSON d'una recepta:


```
{
  "id": 1,
  "name": "Classic Margherita Pizza",
  "ingredients": [
    "Pizza dough",
    "Tomato sauce",
    "Fresh mozzarella cheese",
    "Fresh basil leaves",
    "Olive oil",
    "Salt and pepper to taste"
  ],
  "instructions": [
    "Preheat the oven to 475°F (245°C).",
    "Roll out the pizza dough and spread tomato sauce evenly.",
    "Top with slices of fresh mozzarella and fresh basil leaves.",
    "Drizzle with olive oil and season with salt and pepper.",
    "Bake in the preheated oven for 12-15 minutes or until the crust is golden brown.",
    "Slice and serve hot."
  ],
  "prepTimeMinutes": 20,
  "cookTimeMinutes": 15,
  "servings": 4,
  "difficulty": "Easy",
  "cuisine": "Italian",
  "caloriesPerServing": 300,
  "tags": ["Pizza", "Italian"],
  "userId": 166,
  "image": "https://cdn.dummyjson.com/recipe-images/1.webp",
  "rating": 4.6,
  "reviewCount": 98,
  "mealType": ["Dinner"]
}
```