Switch back to the `main` branch:

```
(venv)$ git checkout main
```

Now that you are back to the original version of the application, let's incorporate MongoDB as the database platform and implement CRUD operations in the next section.

Setting up MongoDB

There are a number of libraries that allow us to integrate MongoDB into our FastAPI application. However, we'll be using **Beanie**, an asynchronous **Object Document Mapper** (**ODM**) library, to execute database operations from our application.

Let's install the `beanie` library by running the following command:

```
(venv)$ pip install beanie
```

Before diving into the integration, let's look at some of the methods from the Beanie library and also how database tables are created in this section.

Document

In SQL, the data stored in rows and columns are contained in the table. In a NoSQL database, it is called a document. The document represents how the data will be stored in the database collection. Documents are defined the same way a Pydantic model is defined, except that the `Document` class from the Beanie library is inherited instead.

An example document is defined as follows:

```
from beanie import Document

class Event(Document):
    name: str
    location: str

    class Settings:
        name = "events"
```

The `Settings` subclass is defined to tell the library to create the collection name passed in the MongoDB database.

Now that we know how to create a document, let's look at the methods used for carrying out CRUD operations:

- `.insert()` and `.create()`: The `.insert()` and `.create()` methods are called by the document instance to create a new record in the database. You can also choose to use the `.insert_one()` method to add a singular entry to the database.

  To insert many entries into the database, the `.insert_many()` method, which takes a list of the document instance, is called, such as the following:

  ```
  event = Event(name="Packt office launch",
  location="Hybrid")
  await event.create()
  await Event.insert_one(event)
  ```

- `.find()` and `.get()`: The `.find()` method is used to find a list of documents matching the search criteria passed as the method argument. The `.get()` method is used to retrieve a single document matching the supplied ID. A single document matching a search criterion can be found using the `.find_one()` method, such as the following:

  ```
  event = await Event.get("74478287284ff")
  event = await Event.find(Event.location == "Hybrid").to_
  list() # Returns a list of matching items
  event = await.find_one(Event.location == "Hybrid") #
  Returns a single event
  ```

- `.save()`, `.update()`, and `.upsert()`: To update a document, any of these methods can be used. The `.update()` method takes an update query, and the `.upsert()` method is used when a document doesn't match the search criteria. In this chapter, we'll be making use of the .`update()` method. An update query is an instruction followed by the MongoDB database, such as the following:

  ```
  event = await Event.get("74478287284ff")
  update_query = {"$set": {"location": "virtual"}}
  await event.update(update_query)
  ```

  In this code block, we first retrieve the event and then create an update query to set the `location` field in the event collection to `virtual`.

- `.delete()`: This method is responsible for removing a document record from the database, such as the following:

  ```
  event = await Event.get("74478287284ff")
  await event.delete()
  ```

Now that we have learned how the methods contained in the Beanie library work, let's initialize the database in our event planner application, define our documents, and implement the CRUD operations.

Initializing the database

Let's look at the steps to do this:

1. In the database folder, create a `connection.py` file:

   ```
   (venv)$ touch connection.py
   ```

   Pydantic enables us to read environment variables by creating a child class of the `BaseSettings` parent class. When building web APIs, it is a standard practice to store configuration variables in an environment file.

2. In `connection.py`, add the following:

   ```
   from beanie import init_beanie
   from motor.motor_asyncio import AsyncIOMotorClient
   from typing import Optional
   from pydantic import BaseSettings

   class Settings(BaseSettings):
       DATABASE_URL: Optional[str] = None

       async def initialize_database(self):
           client = AsyncIOMotorClient(self.DATABASE_URL)
           await init_beanie(
           database=client.get_default_database(),
                         document_models=[])

       class Config:
           env_file = ".env"
   ```

In this code block, we start by importing the dependencies required for initializing the database. Then, we define the `Settings` class, which has a `DATABASE_URL` value that is read from the `env_file` environment defined in the `Config` subclass. We also define an `initialize_database` method to initialize the database.

The `init_beanie` method takes the database client, which is the mongo version of the engine created in the SQLModel section, and a list of documents.

3. Let's update the model files in the models directory to include the MongoDB documents. In `models/events.py`, replace the contents with the following:

```
from beanie import Document
from typing import Optional, List

class Event(Document):
    title: str
    image: str
    description: str
    tags: List[str]
    location: str

    class Config:
        schema_extra = {
            "example": {
                "title": "FastAPI Book Launch",
                "image": "https:
                //linktomyimage.com/image.png",
                "description": "We will be discussing
                the contents of the FastAPI book in
                this event. Ensure to come with your
                own copy to win gifts!",
                "tags": ["python", "fastapi", "book",
                "launch"],
                "location": "Google Meet"
            }
        }
    class Settings:
        name = "events"
```

4. Let's create a Pydantic model for the `UPDATE` operations:

```
class EventUpdate(BaseModel):
    title: Optional[str]
    image: Optional[str]
    description: Optional[str]
    tags: Optional[List[str]]
    location: Optional[str]

    class Config:
        schema_extra = {
            "example": {
                "title": "FastAPI Book Launch",
                "image": "https:
                //linktomyimage.com/image.png",
                "description": "We will be discussing
                the contents of the FastAPI book in
                this event. Ensure to come with your
                own copy to win gifts!",
                "tags": ["python", "fastapi", "book",
                "launch"],
                "location": "Google Meet"
            }
        }
```

5. In `model/users.py`, replace the content of the file with the following:

```
from typing import Optional, List
from beanie import Document, Link

from pydantic import BaseModel, EmailStr

from models.events import Event
```