Part 2: Building and Securing FastAPI Applications

Upon completing this part, you will be able to build a fully functional and secure application with FastAPI. This part uses the knowledge from the previous part and buttresses it into building a more functional application with higher complexity than the application built in the second chapter. You will also be able to integrate and connect to a SQL and NoSQL (MongoDB) database as well as being able to secure a FastAPI application by the end of this part.

This part comprises the following chapters:

- *Chapter 5*, *Structuring FastAPI Applications*
- *Chapter 6*, *Connecting to a Database*
- *Chapter 7*, *Securing FastAPI Applications*

5
Structuring FastAPI Applications

In the last four chapters, we looked at the basic steps involved in understanding FastAPI and creating a FastAPI application. The application that we have built so far is a single-file todo application that demonstrates the flexibility and power of FastAPI. The key takeaway from the preceding chapters is how easy it is to build an application using FastAPI. However, there is a need for proper structuring of an application with increased complexity and functionalities.

Structuring refers to the arrangement of application components in an organized format, which can be modular to improve the readability of the application's code and content. An application with proper structuring enables faster development, faster debugging, and an overall increase in productivity.

By the end of this chapter, you will be equipped with the knowledge of what structuring is and how to structure your API. In this chapter, you'll be covering the following topics:

- Structuring application routes and models
- Implementing models for a planner API

Technical requirements

The code used in this chapter can be found at `https://github.com/PacktPublishing/Building-Python-Web-APIs-with-FastAPI/tree/main/ch05/planner`.

Structuring in FastAPI applications

For this chapter, we'll be building an event planner. Let's design the application structure to look like this:

```
planner/
  main.py
  database/
    __init__.py
    connection.py
  routes/
    __init__.py
    events.py
    users.py
  models/
    __init__.py
    events.py
    users.py
```

The first step is to create a new folder for the application. It will be named `planner`:

```
$ mkdir planner && cd planner
```

In the newly created `planner` folder, create an entry file, `main.py`, and three subfolders – `database`, `routes`, and `models`:

```
$ touch main.py
$ mkdir database routes models
```

Next, create `__init__.py` in every folder:

```
$ touch {database,routes,models}/__init__.py
```

In the `database` folder, let's create a blank file, `database.py`, which will handle the database abstractions and configurations we'll be using in the next chapter:

```
$ touch database/connection.py
```

In both the `routes` and `models` folders, we'll create two files, `events.py` and `users.py`:

```
$ touch {routes,models}/{events,users}.py
```

Each file has its function, as stated here:

- Files in the `routes` folder:
  - `events.py`: This file will handle routing operations such as creating, updating, and deleting events.
  - `users.py`: This file will handle routing operations such as the registration and signing-in of users.
- Files in the `models` folder:
  - `events.py`: This file will contain the model definition for events operations.
  - `users.py`: This file will contain the model definition for user operations.

Now that we have successfully structured our API and grouped similar files with respect to their functions into components, let's begin the implementation of the application in the next section.

Building an event planner application

In this section, we'll be building an event planner application. In this application, registered users will be able to create, update, and delete events. Events created can be viewed by navigating to the event page created automatically by the application.

Each registered user and event will have a unique ID. This is to prevent conflict in managing users and events with the same ID. In this section, we will not be prioritizing authentication or database management, as this will be discussed in depth in *Chapter 6*, *Connecting to a Database*, and *Chapter 7*, *Securing FastAPI Applications*.