To kick-start the development, let us create a virtual environment and activate it in our project directory:

```
$ python3 -m venv venv
$ source venv/bin/activate
```

Next, let's install the application dependencies:

```
(venv)$ pip install fastapi uvicorn "pydantic[email]"
```

Lastly, save the requirements into `requirements.txt`:

```
(venv)$ pip freeze > requirements.txt
```

Now that we have successfully installed our dependencies and set up our development environment, let's implement the application's models next.

Implementing the models

Let's look at the steps for implementing our model:

1. The first step in building our application is to define the models for the event and user. The models describe how data will be stored, inputted, and represented in our application. The following diagram shows the modeling for both the user and event as well as their relationship:

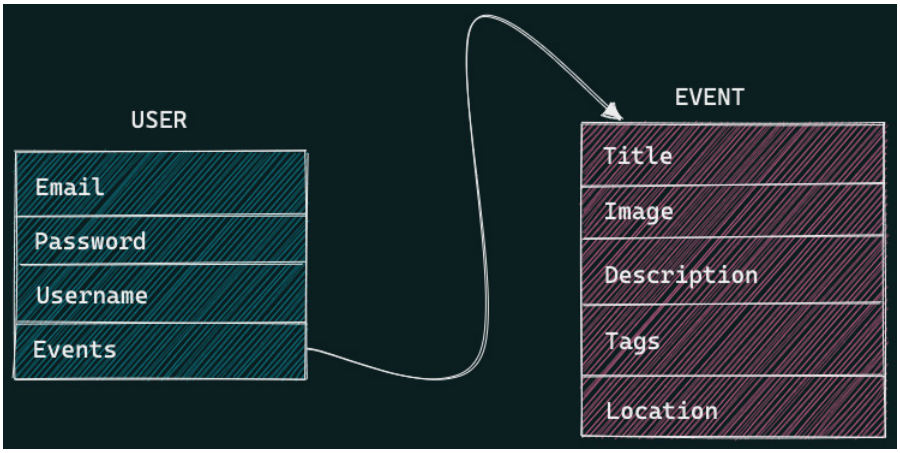

Figure 5.1 – The model flow for the user and event

As shown in the previous model diagram, each user will have an **Events** field, which is a list of the events they have ownership of.

2. Let's define the `Event` model in `models/events.py`:

```
from pydantic import BaseModel
from typing import List

class Event(BaseModel):
    id: int
    title: str
    image: str
    description: str
    tags: List[str]
    location: str
```

3. Let's define a `Config` subclass under the `Event` class to show an example of what the model data will look like when we visit the documentation:

```
    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"
            }
        }
```

Our event model in the first block of code contains five fields:

- The event title
- A link to the event image banner
- The description of the event
- Event tags for grouping
- The location of the event

In the second block of code, we define example event data. This is aimed at guiding us when creating a new event from our API.

4. Now that we have our event model defined, let's define the `User` model:

```
from pydantic import BaseModel, EmailStr
from typing import Optional, List
from models.events import Event

class User(BaseModel):
    email: EmailStr
    password: str
    events: Optional[List[Event]]
```

Our `User` model defined previously contains the following fields:

- The user's email
- The user's password
- A list of events created by the user, which is empty by default

5. Now that we have defined our `User` model, let's create an example that indicates how the user data is stored and set:

```
    class Config:
        schema_extra = {
            "example": {
                "email": fastapi@packt.com,
                "username": "strong!!!",
                "events": [],
           }
        }
```

6. Next, we'll create a new model, `NewUser`, which inherits from the `User` model; this new model will be used as the data type when registering a new user. The `User` model will be used as response models where we do not want to interact with the password, reducing the amount of work to be done.
7. Lastly, let's implement a model for signing users in:

```
class UserSignIn(BaseModel):
    email: EmailStr
    password: str

    class Config:
        schema_extra = {
            "example": {
                "email": fastapi@packt.com,
                "password": "strong!!!",
                "events": [],
           }
        }
```

Now that we have successfully implemented our models, let's implement the routes in the next section.

Implementing routes

The next step in building our application is to set up the routing system of our API. We'll be designing the routing system for events and users. The user route will consist of sign-in, sign-out, and sign-up routes. An authenticated user will have access to the routes for creating, updating, and deleting an event, while the public can view the event once it has been created. The following diagram shows the relationship between both routes:

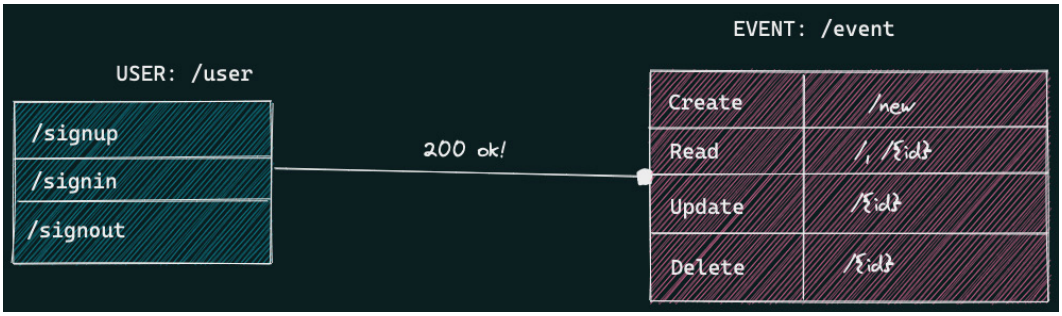

Figure 5.2 – The routes for user and event operations

Let's look at both routes in detail next.

User routes

Now that we have a clear idea of what routes to implement from *Figure 5.2*, we'll start by defining the user routes in `users.py`. Let's look at the steps:

1. Start by defining a basic sign-up route:

```
from fastapi import APIRouter, HTTPException, status
from models.user import User, UserSignIn

user_router = APIRouter(
    tags=["User"]
)

users = {}

@user_router.post("/signup")
async def sign_new_user(data: NewUser) -> dict:
    if data.email in users:
        raise HTTPException(
```