```
            status_code=status.HTTP_409_CONFLICT,
            detail="User with supplied username
            exists"
        )
    users[data.email] = data
    return {
        "message": "User successfully registered!"
    }
```

In the sign-up route defined previously, we're making use of an in-app database. (We'll be introducing a database in *Chapter 6*, *Connecting to a Database*.)

The route checks whether a user with a similar email address exists in the database before adding a new one.

2. Let's implement the sign-in route:

```
@user_router.post("/signin")
async def sign_user_in(user: UserSignIn) -> dict:
    if users[user.email] not in users:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND
            detail="User does not exist"
        )
    if users[user.email].password != user.password:
            raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN
            detail="Wrong credentials passed"
        )

    return {
        "message": "User signed in successfully"
    }
```

In this route, the first step taken is to check whether such a user exists in the database, and if the user doesn't exist, an exception is raised. If the user exists, the application proceeds to check whether the passwords match before returning a successful message or an exception.

In our routes, we're storing the passwords plainly without any encryption. This is used only for demonstration purposes and is a wrong practice in software engineering in general. Proper storage mechanisms such as encryption will be discussed in *Chapter 6*, *Connecting to a Database*, where our application will move from an in-app database to a real database.

3. Now that we have defined the routes for the user operations, let's register them in `main.py` and start our application. Let's start by importing our libraries and the user routes definition:

```
from fastapi import FastAPI
from routes.user import user_router

import uvicorn
```

4. Next, let's create a `FastAPI` instance and register the route and the application:

```
app = FastAPI()

# Register routes

app.include_router(user_router, prefix="/user")

if __name__ == "__main__":
    uvicorn.run("main:app", host="0.0.0.0", port=8080,
    reload=True)
```

In this block of code, we have created an instance of `FastAPI` and registered the route.

5. Next, we make use of the `uvicorn.run()` method to start our application on port `8080` and set the hot reload to `True`. In the terminal, start the application:

```
(venv)$ python main.py
INFO:     Will watch for changes in these directories:
['/Users/youngestdev/Work/Building-Web-APIs-with-FastAPI-
and-Python/ch05/planner']
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press
CTRL+C to quit)
INFO:     Started reloader process [6547] using
statreload
```

```
INFO:     Started server process [6550]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

6. Now that our application has started successfully, let's test the user routes we implemented. We'll start by signing up a user:

```
(venv)$ curl -X 'POST' \
  'http://0.0.0.0:8080/user/signup' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "email": "fastapi@packt.com",
  "password": "Stro0ng!",
  "username": "FastPackt"
}'
```

The preceding request returns a response:

```
{
  "message": "User successfully registered!"
}
```

7. The preceding response indicates the success of the operation performed. Let's test the sign-in route:

```
(venv)$ curl -X 'POST' \
  'http://0.0.0.0:8080/user/signin' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "email": "fastapi@packt.com",
  "password": "Stro0ng!"
}'
```

The preceding response to the request is as follows:

```
{
  "message": "User signed in successfully"
}
```

8. If we pass a wrong password, our application should return a different message:

```
(venv)$ curl -X 'POST' \
  'http://0.0.0.0:8080/user/signin' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "email": "fastapi@packt.com",
  "password": "password!"
}'
```

The response from the preceding request is as follows:

```
{
  "detail": "Wrong credential passed"
}
```

We can also view our routes from the interactive documentation provided by FastAPI, which is powered by Swagger. Let's visit `http://0.0.0.0:8080/docs` in our browser to gain access to the interactive documentation:

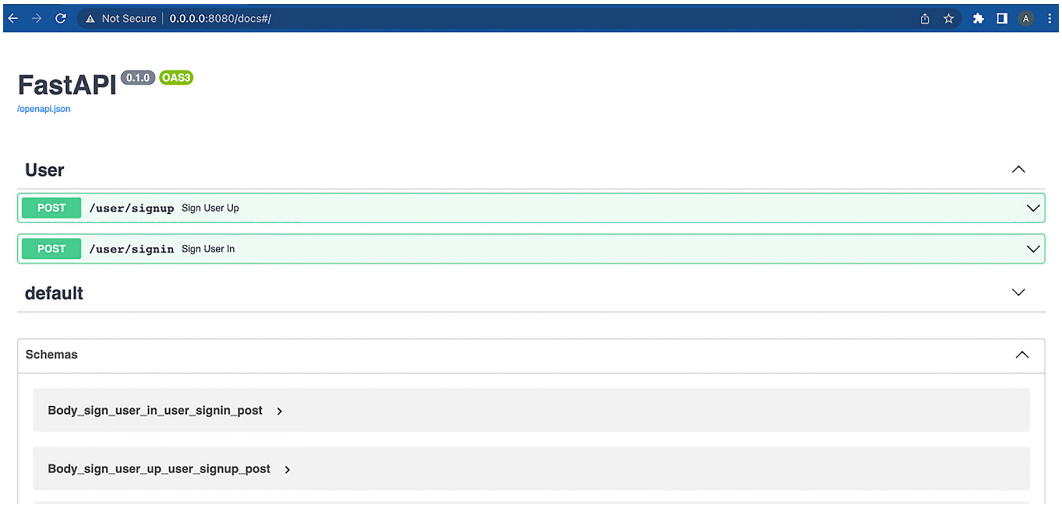

Figure 5.3 – The planner application viewed on the interactive documentation page

Now that we have successfully implemented the user routes, let's implement the routes for event operations in the next section.

Events routes

With the user routes in place, the next step is to implement the routes for event operations. Let's look at the steps:

1. Start by importing the dependencies and defining the event router:

```
from fastapi import APIRouter, Body, HTTPException,
status
from models.events import Event
from typing import List

event_router = APIRouter(
    tags=["Events"]
)

events = []
```

2. The next step is to define the route to retrieve all the events and an event matching a supplied ID in the database:

```
@event_router.get("/", response_model=List[Event])
async def retrieve_all_events() -> List[Event]:
    return events

@event_router.get("/{id}", response_model=Event)
async def retrieve_event(id: int) -> Event:
    for event in events:
        if event.id == id:
            return event
    raise HTTPException(
        status_code=status. HTTP_404_NOT_FOUND,
        detail="Event with supplied ID does not exist"
    )
```

In the second route, we're raising an `HTTP_404_NOT_FOUND` exception when an event with the supplied ID doesn't exist.