```
    user.username)
    ..
    if hash_password.verify_hash(user.password,
    user_exist.password):
        access_token = create_access_token(
        user_exist.email)
        return {
            "access_token": access_token,
            "token_type": "Bearer"
        }
```

In the preceding code block, we have injected the `OAuth2PasswordRequestForm` class as the dependency for this function, ensuring the OAuth spec is strictly followed. In the function body, we compare the password and return an access token and a token type. Before we test the updated route, let's create a response model for the login route in `models/users.py` to replace the `UserSignIn` model class, which isn't used anymore:

```
class TokenResponse(BaseModel):
    access_token: str
    token_type: str
```

Update the imports and the response model for the sign-in route:

```
from models.users import User, TokenResponse

@user_router.post("/signin", response_model=TokenResponse)
```

Let's visit the interactive docs to confirm that the request body is compliant with the OAuth2 specs at `http://0.0.0.0:8080/docs`:

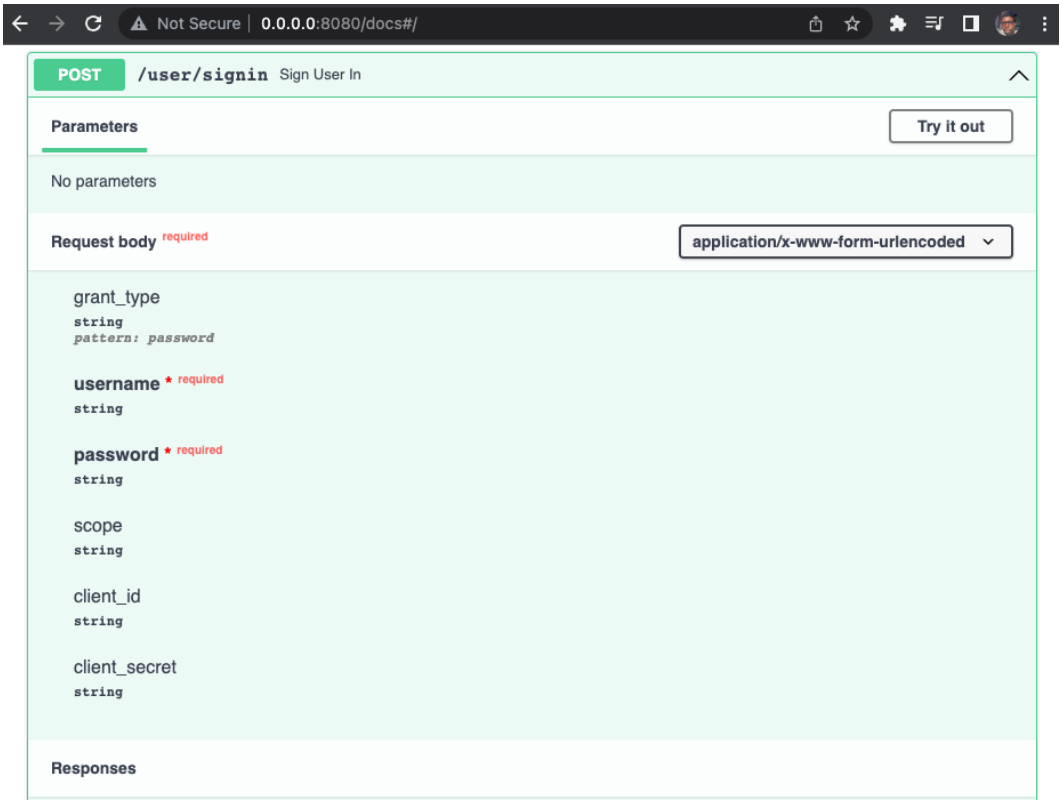

Figure 7.5 – Request body for updated sign-in route

Let's sign in to verify that the route works properly:

```
$ curl -X 'POST' \
  'http://0.0.0.0:8080/user/signin' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -d 'grant_type=&username=reader%40packt.
com&password=exemplary&scope=&client_id=&client_secret='
```

The response returned is an access token and the token type:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2
VyIjoicmVhZGVyQHBhY2t0LmNvbSIsImV4cGlyZXMiOjE2NTA4Mjc0MjQuMDg2
NDAxfQ.LY4i5EjIzlsKdfMyWKi7XH7lLeDuVt3832hNfkQx8C8",
  "token_type": "Bearer"
}
```

Now that we have confirmed that the route works as expected, let's update the event routes to allow only authorized users' **CREATE**, **UPDATE**, and **DELETE** events.

Updating event routes

Now that we have our authentication in place, let's inject the authentication dependency into the `POST`, `PUT`, and `DELETE` route functions:

```
from auth.authenticate import authenticate


async def create_event(body: Event, user: str =
Depends(authenticate)) -> dict:
..

async def update_event(id: PydanticObjectId, body: EventUpdate,
user: str = Depends(authenticate)) -> Event:
..

async def delete_event(id: PydanticObjectId, user: str =
Depends(authenticate)) -> dict:
..
```

With the dependencies injected, the interactive docs website is automatically updated to show protected routes. If we log on to `http://0.0.0.0:8080/docs`, we can see the **Authorize** button at the top right and the padlocks on the event routes:

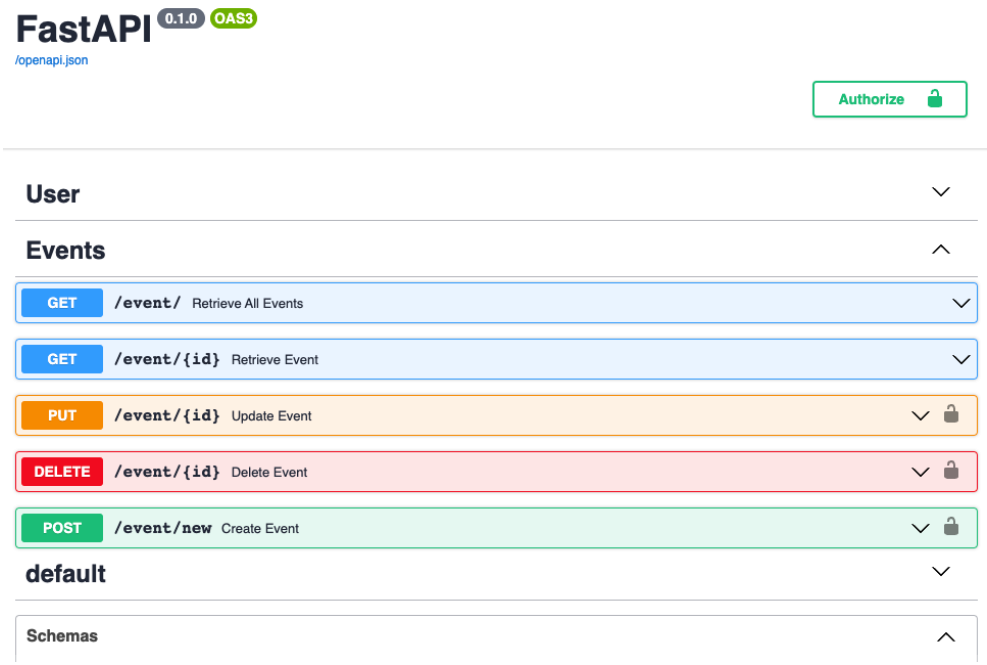

Figure 7.6 – Updated documentation page

If we click on the **Authorize** button, a sign-in modal is displayed. Inputting our credentials and password returns the following screen:

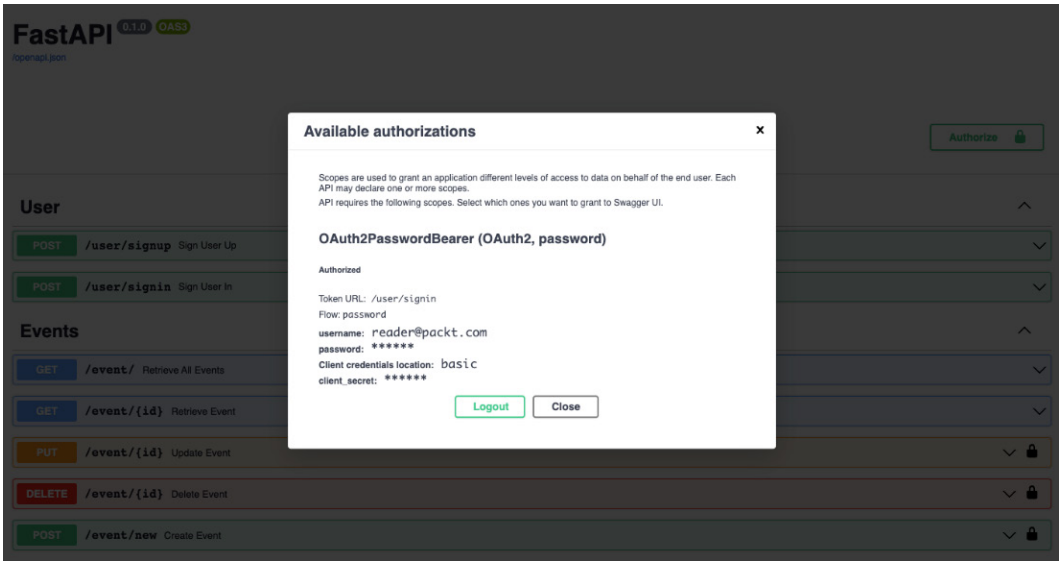

Figure 7.7 – Authenticated user

Now that we have successfully signed in, we can create an event:

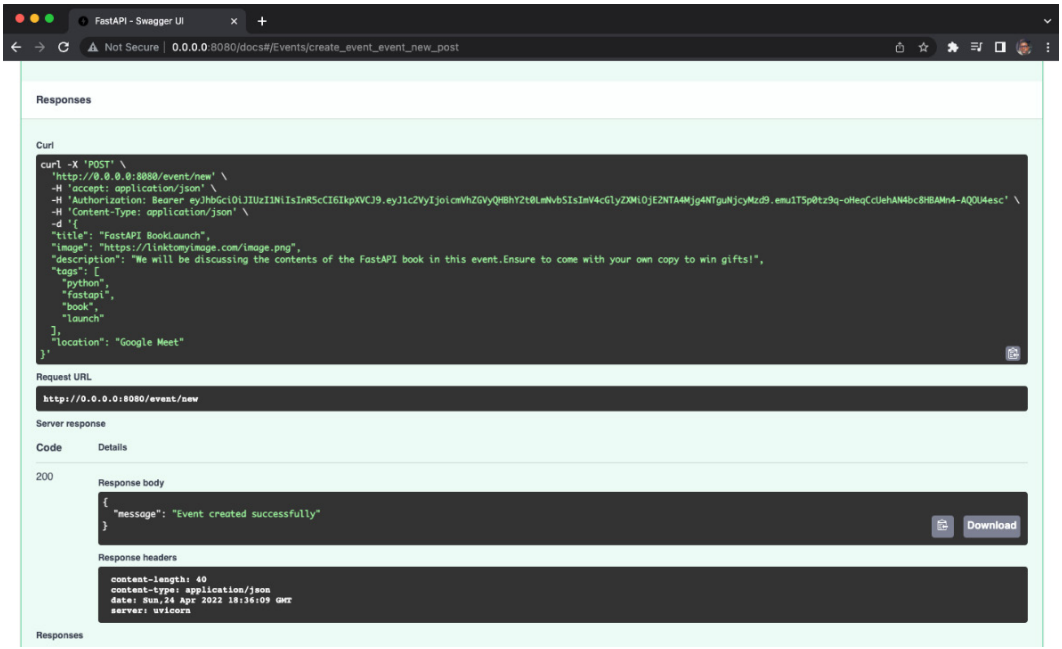

Figure 7.8 – Create a new event

The same operations can be performed from the command line. First, let's get our access token:

```
$ curl -X 'POST' \
  'http://0.0.0.0:8080/user/signin' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -d 'grant_type=&username=reader%40packt.
com&password=exemplary&scope=&client_id=&client_secret='
```

The request sent returns the access token, which is a JWT string, and the token type, which is of type `Bearer`:

```
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ1c2VyIjoicmVhZGVyQHBhY2t0LmNvbSIsImV4cGlyZXMiOjE2NTA4MjkxOD
MuNTg3NjAyfQ.MOXjI5GXnyzGNftdlxDGyM119_L11uPq8yCxBHepf04",
  "token_type": "Bearer"
}
```