FastAPI also provides a `Path` class that distinguishes path parameters from other arguments present in the route function. The `Path` class also helps give route parameters more context during the documentation automatically provided by OpenAPI via **Swagger** and **ReDoc** and acts as a validator.

Let's modify the route definition:

```
from fastAPI import APIRouter, Path

from model import Todo

todo_router = APIRouter()

todo_list = []


@todo_router.post("/todo")
async def add_todo(todo: Todo) -> dict:
   todo_list.append(todo)
   return {
       "message": "Todo added successfully."
   }


@todo_router.get("/todo")
async def retrieve_todo() -> dict:
   return {
       "todos": todo_list
   }


@todo_router.get("/todo/{todo_id}")
async def get_single_todo(todo_id: int = Path(..., title="The
ID of the todo to retrieve")) -> dict:
   for todo in todo_list:
       if todo.id == todo_id:
           return {
               "todo": todo
```

```
            }
    return {
        "message": "Todo with supplied ID doesn't exist."
    }
```

Tip – Path(..., kwargs)

The `Path` class takes a first positional argument set to `None` or ellipsis (...). If the first argument is set to an ellipsis (...), the path parameter becomes required. The `Path` class also contains arguments used for numerical validations if a path parameter is a number. Definitions include `gt` and `le` – `gt` means greater than and `le` means less than. When used, the route will validate the path parameter against these arguments.

Query parameters

A query parameter is an optional parameter that usually appears after a question mark in a URL. It is used to filter requests and return specific data based on the queries supplied.

In a route handler function, an argument that isn't homonymous with the path parameter is a query. You can also define a query by creating an instance of the FastAPI `Query()` class in the function argument, such as the following:

```
async query_route(query: str = Query(None):
    return query
```

We will be looking at the use cases of the query parameters later on in the book when we discuss how to build more advanced applications than a todo application.

Now that you have learned how to create routes, validate request bodies, and use path and query parameters in your FastAPI application, you will learn how these components work hand in hand to form a request body in the next section.

Request body

In the previous sections, we learned how to use the `APIRouter` class and Pydantic models for request body validations and discussed path and query parameters.

A request body is data that you send to your API using a routing method such as `POST` and `UPDATE`.

POST and UPDATE

The `POST` method is used when an insertion into the server is to be made, and the `UPDATE` method is used when existing data in the server is to be updated.

Let's take a look at a `POST` request from earlier on in the chapter:

```
(venv)$ curl -X 'POST' \
  'http://127.0.0.1:8000/todo' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": 2,
  "item": "Validation models help with input types"
}'
```

In the preceding request, the request body is as follows:

```
{
  "id": 2,
  "item": "Validation models help with input types.."
}
```

Tip

FastAPI also provides us with a `Body()` class to provide extra validation.

We have learned about models in FastAPI. They also serve an additional purpose in documenting our API endpoints and request body types. In the next subsection, we will learn about the documentation pages generated by default in FastAPI applications.

FastAPI Automatic Docs

FastAPI generates JSON schema definitions for our models and automatically documents our routes, including their request body type, path and query parameters, and response models. This documentation is of two types:

- **Swagger**
- **ReDoc**

Swagger

The documentation hosted by swagger provides an interactive environment to test our API. You can access it by appending `/docs` to the application address. In your web browser, visit the `http://127.0.0.1:8000/docs` URL:

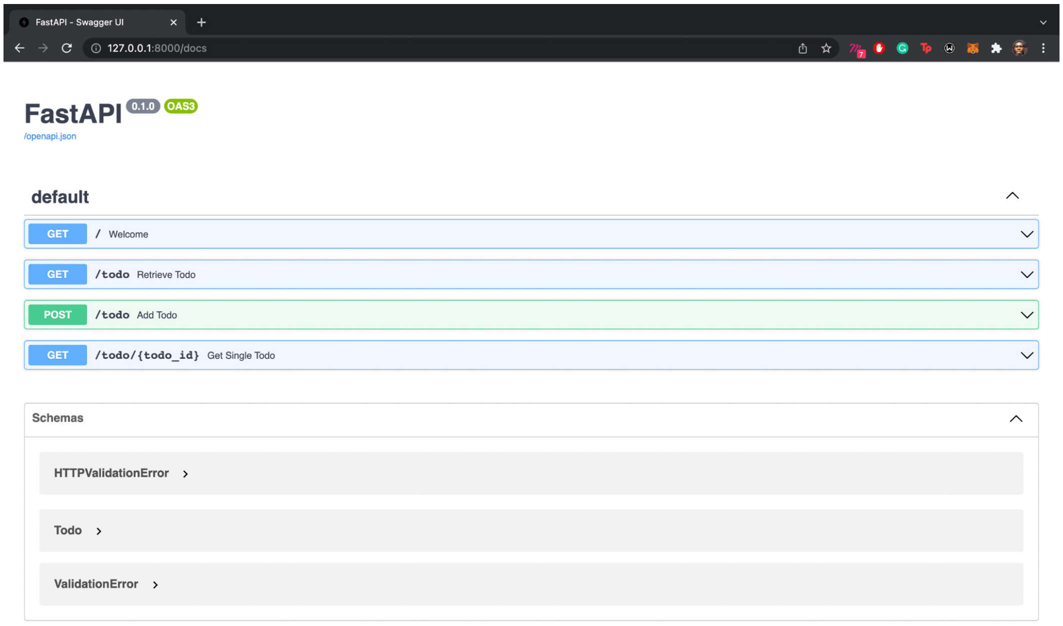
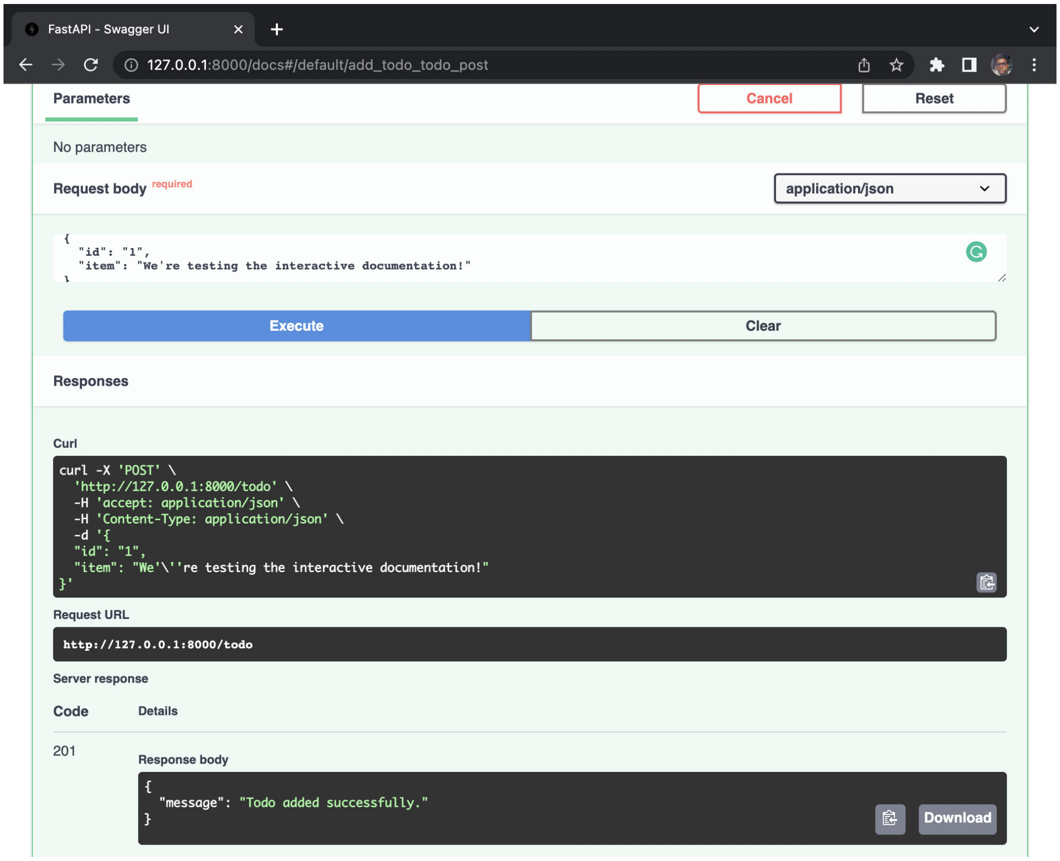


Figure 2.1 – The FastAPI interactive documentation

The interactive documentation allows us to test our methods. Let's add a todo from the interactive documentation:



```
{
  "id": "1",
  "item": "We're testing the interactive documentation!"
}
```

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/todo' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": "1",
  "item": "We'\''re testing the interactive documentation!"
}'
```

Request URL

```
http://127.0.0.1:8000/todo
```

Server response

Code	Details
201	Response body

```
{
  "message": "Todo added successfully."
}
```

Figure 2.2 – Route test from interactive documentation

Now that we know what the interactive documentation looks like, let's check the documentation generated by ReDoc.