ReDoc

The ReDoc documentation gives a more detailed and direct presentation of the models, routes, and API. You can access it by appending `/redoc` to the application address. In your web browser, visit the `http://127.0.0.1:8000/redoc` URL:

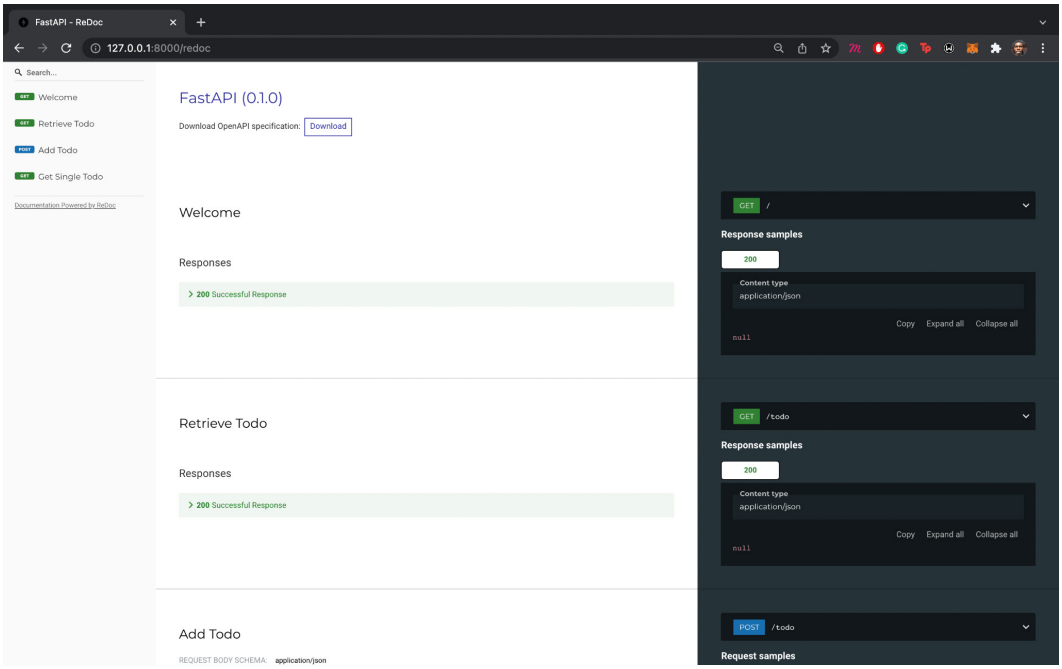

Figure 2.3 – The ReDoc-powered documentation portal

To correctly generate **JSON Schema**, you can set examples of how a user will fill data in the model. An example is set by embedding a `Config` class into a model class. Let's add an example schema in our `Todo` model:

```
class Todo(BaseModel):
    id: int
    item: str

    class Config:
        Schema_extra = {
            "Example": {
                "id": 1,
```

```
                "item": "Example schema!"
                        }
        }
```

Refresh the documentation page for ReDoc and click on **Add Todo** on the left pane. The example is shown in the right pane:

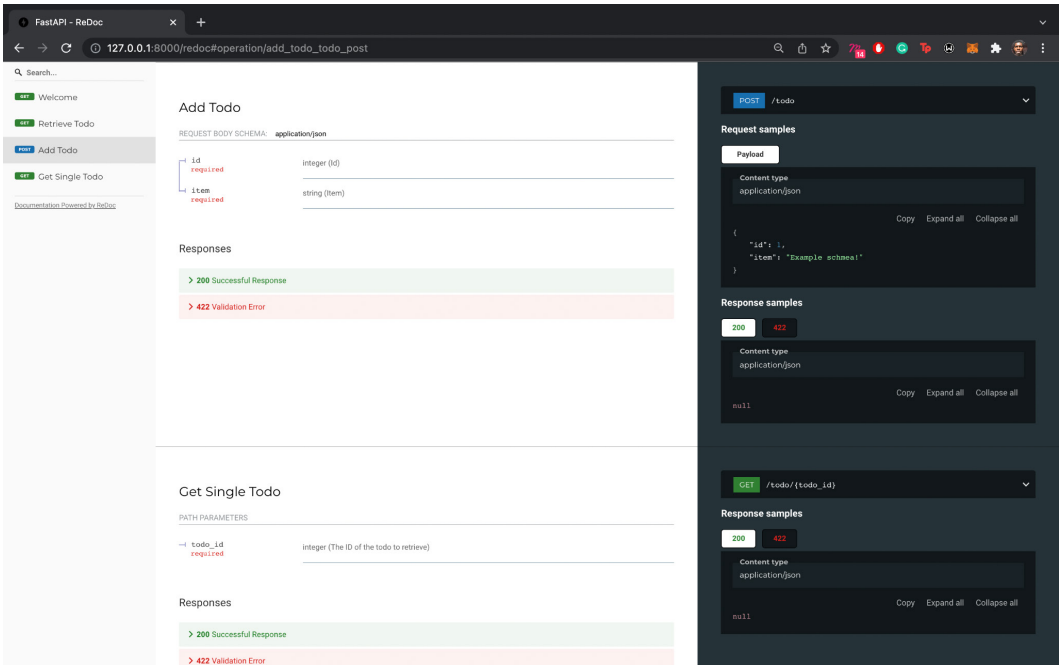

Figure 2.4 – The documentation portal shows example schema

Also in the interactive documentation, the example schema can be seen:

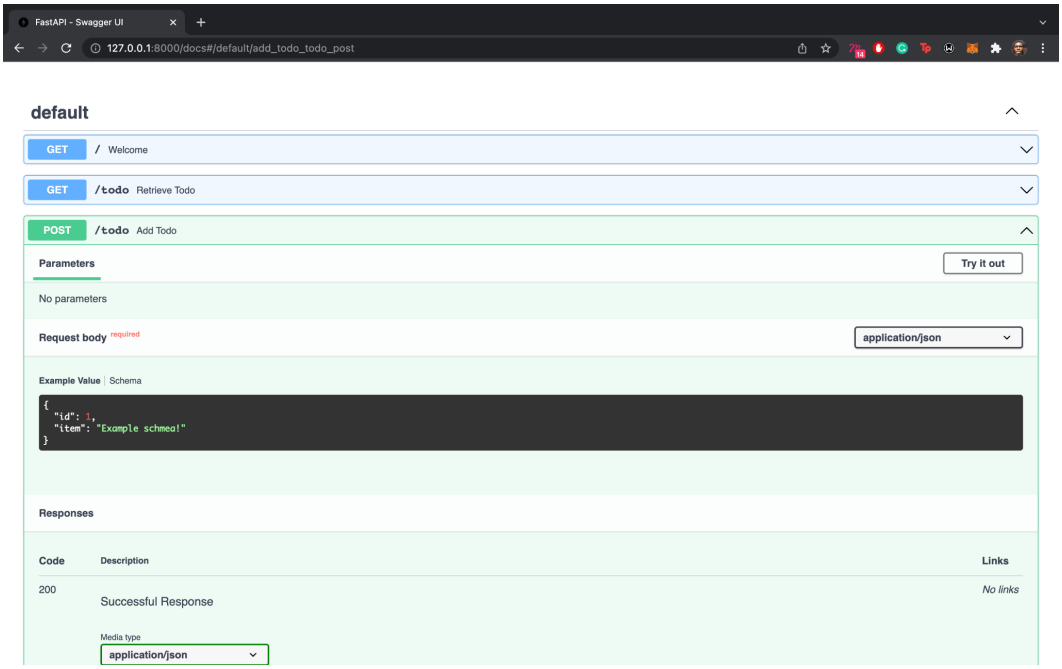

Figure 2.5 – The documentation portal shows example schema

We have learned how to add example schema data to guide users on how to send requests to the API and test the application from Swagger's interactive documentation. The documentation provided by ReDoc isn't left out, as it serves as a knowledge base on how to use the API.

Now that we have learned what the `APIRouter` class is and how to use it, the request body, path and query parameters, and validating request bodies with Pydantic models, let's update our todo app to include routes for updating and deleting a todo item.

Building a simple CRUD app

We have built routes for **creating** and **retrieving** todos. Let's build the routes for **updating** and **deleting** the added todo. Let's start by creating a model for the request body for the UPDATE route in `model.py`:

```
class TodoItem(BaseModel):
    item: str

    class Config:
        schema_extra = {
            "example": {
                "item": "Read the next chapter of the book"
            }
        }
```

Next, let's write the route for updating a todo in `todo.py`:

```
from fastapi import APIRouter, Path
from model import Todo, TodoItem

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."
    }


@todo_router.put("/todo/{todo_id}")
async def update_todo(todo_data: TodoItem, todo_id: int =
Path(..., title="The ID of the todo to be updated")) -> dict:
    for todo in todo_list:
        if todo.id == todo_id:
            todo.item = todo_data.item
            return {
                "message": "Todo updated successfully."
            }
    return {
        "message": "Todo with supplied ID doesn't exist."
    }
```

Let's test the new route. First, let's add a todo:

```
(venv)$ curl -X 'POST' \
  'http://127.0.0.1:8000/todo' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": 1,
```