```
  "item": "Example Schema!"
}'
```

Here is the response:

```
(venv)$ {
  "message": "Todo added successfully."
}
```

Next, let's update the todo by sending a `PUT` request:

```
(venv)$ curl -X 'PUT' \
  'http://127.0.0.1:8000/todo/1' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "item": "Read the next chapter of the book."
}'
```

Here is the response:

```
(venv)$ {
  "message": "Todo updated successfully."
}
```

Let's verify that our todo has indeed been updated:

```
(venv)$ curl -X 'GET' \
  'http://127.0.0.1:8000/todo/1' \
  -H 'accept: application/json'
```

Here is the response:

```
(venv)$ {
  "todo": {
    "id": 1,
    "item": "Read the next chapter of the book"
  }
}
```

From the response returned, we can see that the todo has successfully been updated. Now, let's create the route for deleting a todo and all todos.

In `todo.py`, update the routes:

```
@todo_router.delete("/todo/{todo_id}")
async def delete_single_todo(todo_id: int) -> dict:
   for index in range(len(todo_list)):
       todo = todo_list[index]
       if todo.id == todo_id:
           todo_list.pop(index)
           return {
               "message": "Todo deleted successfully."
           }
   return {
       "message": "Todo with supplied ID doesn't exist."
   }


@todo_router.delete("/todo")
async def delete_all_todo() -> dict:
   todo_list.clear()
   return {
       "message": "Todos deleted successfully."
   }
```

Let's test the `delete` route. First, we 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,
  "item": "Example Schema!"
}'
```

Here is the response:

```
(venv)$ {
  "message": "Todo added successfully."
}
```

Next, delete the todo:

```
(venv)$ curl -X 'DELETE' \
  'http://127.0.0.1:8000/todo/1' \
  -H 'accept: application/json'
```

Here is the response:

```
(venv)$ {
  "message": "Todo deleted successfully."
}
```

Let's verify that the todo has been deleted by sending a `GET` request to retrieve the todo:

```
(venv)$ curl -X 'GET' \
  'http://127.0.0.1:8000/todo/1' \
  -H 'accept: application/json'
```

Here is the response:

```
(venv)$ {
  "message": "Todo with supplied ID doesn't exist.
}
```

In this section, we built a CRUD todo application combining the lessons learned from the preceding sections. By validating our request body, we were able to ensure that proper data is sent to the API. The inclusion of path parameters to our routes also enabled us to retrieve and delete a single todo from our todo list.

Summary

In this chapter, we learned how to use the `APIRouter` class and connect routes defined with it to the primary FastAPI instance. We also learned how to create models for our request bodies and add path and query parameters to our path operations. These models serve as extra validation against improper data types supplied to request body fields. We also built a CRUD todo application to put into practice all that we learned in this chapter.

In the next chapter, you will be introduced to response, response modeling, and error handling in FastAPI. You will first be introduced to the concept of responses and how the knowledge of Pydantic models learned in this chapter helps build models for API responses. You will then learn about status codes and how to use them in your response objects and proper error handling.

3
Response Models and Error Handling

Response models serve as templates for returning data from an API route path. They are built on **Pydantic** to properly render a response from requests sent to the server.

Error handling includes the practices and activities involved in handling errors from an application. These practices include returning adequate error status codes and error messages.

By the end of this chapter, you will know what a response is and what it consists of, and you'll know about error handling and how to handle errors in your FastAPI application. You will also know how to build response models for request responses using Pydantic.

In this chapter, we'll be covering the following topics:

- Responses in FastAPI
- Building a response model
- Error handling