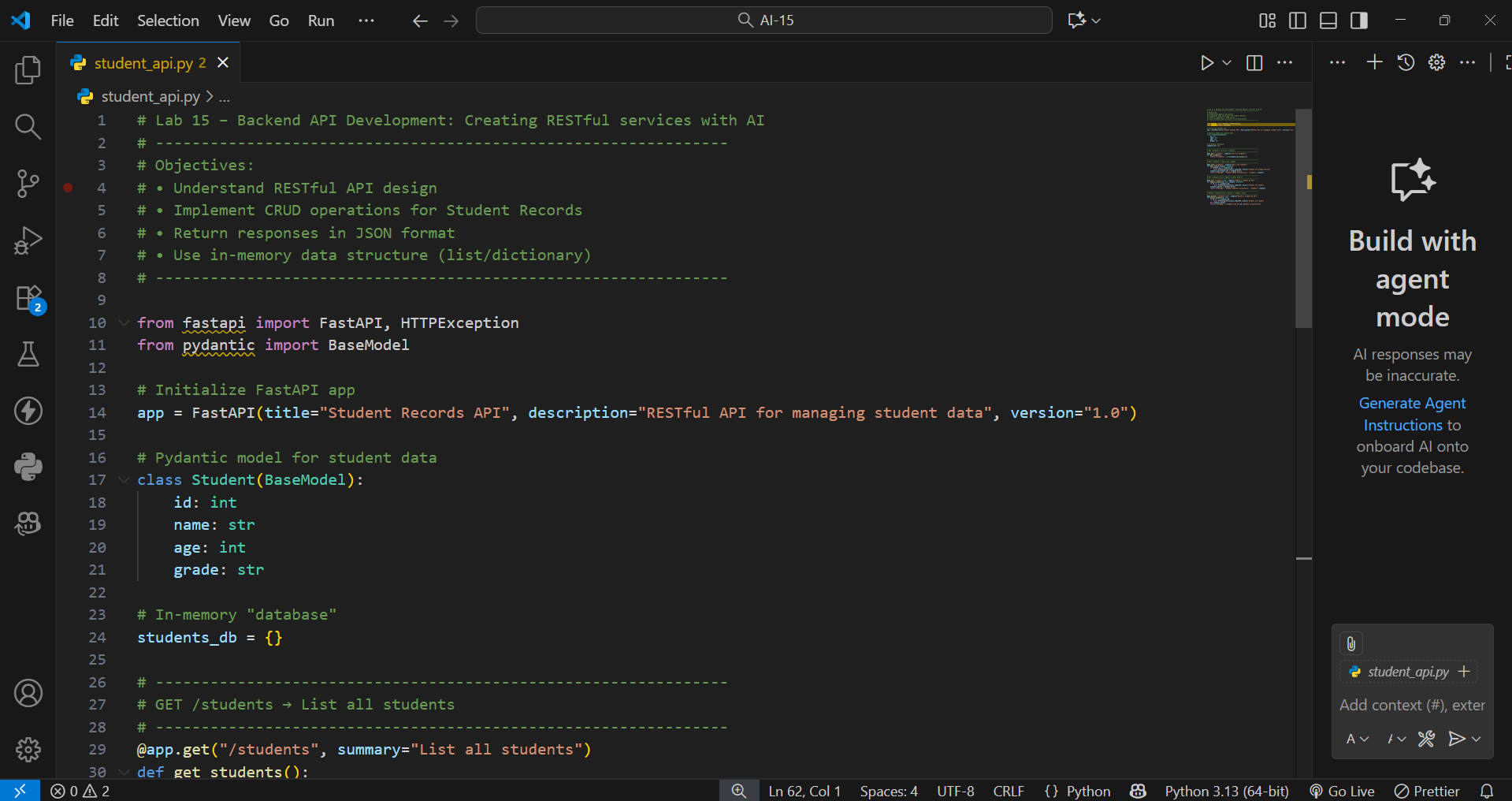
**Assignment-15   
  
HNO:2503A52L16**

**Lab 15 – Backend API Development: Creating RESTful services with AIs**

**Task 1 – Student Records API**

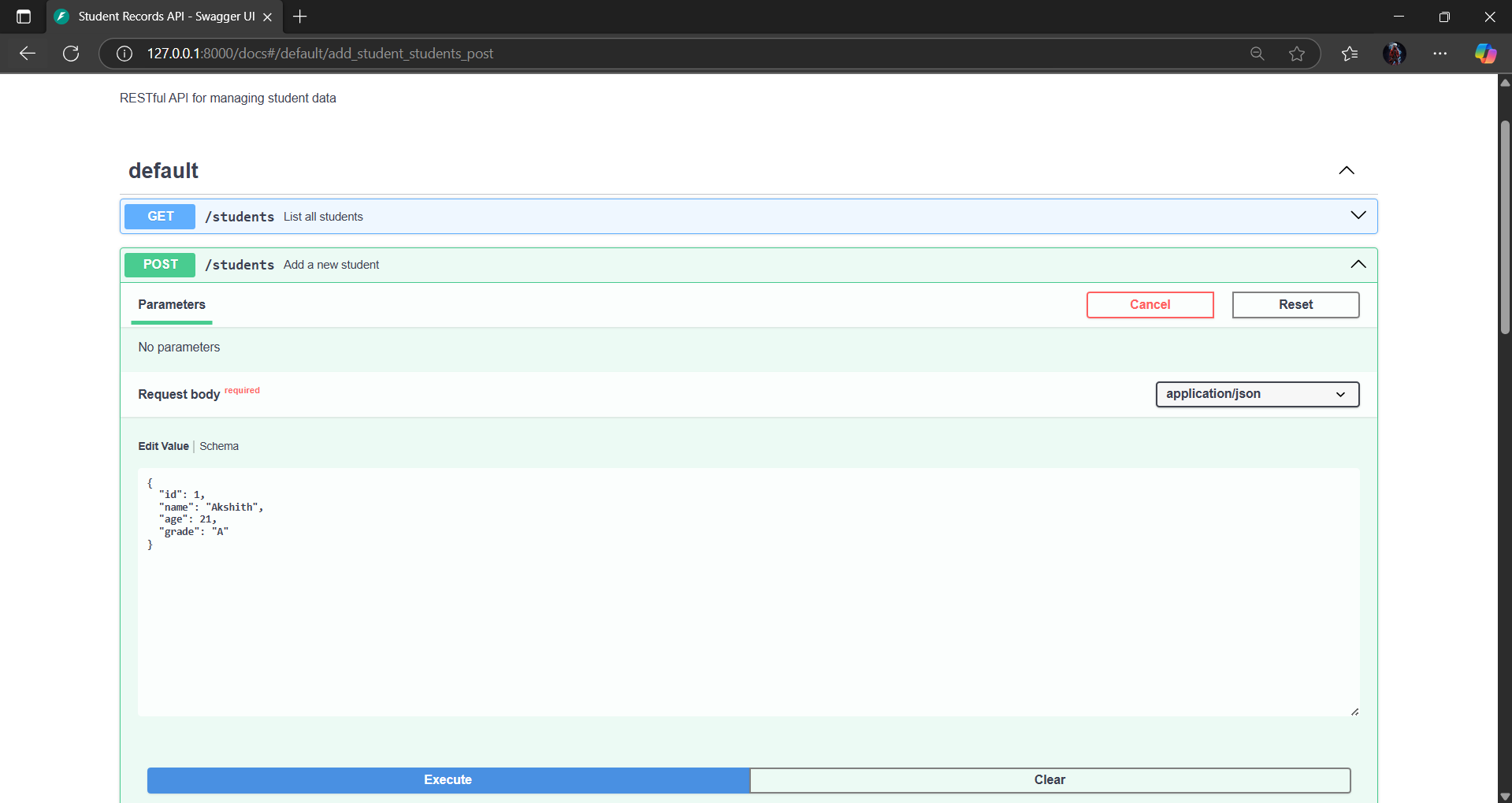
**Prompt:** I have a lab assignment titled **“Lab 15 – Backend API Development: Creating RESTful Services with AI.”** The goal of this lab is to understand the fundamentals of RESTful API design and to use AI-assisted coding tools to build backend services. I need to create a RESTful API that performs **CRUD (Create, Read, Update, Delete)** operations for managing student records. The API should include the following endpoints: GET /students to list all students, POST /students to add a new student, PUT /students/{id} to update student details, and DELETE /students/{id} to delete a student record. The data should be stored using an **in-memory structure** such as a list or dictionary, and all responses must be in **JSON format**. The expected output is a **fully functional API** that demonstrates CRUD operations and includes proper comments and auto-generated documentation.

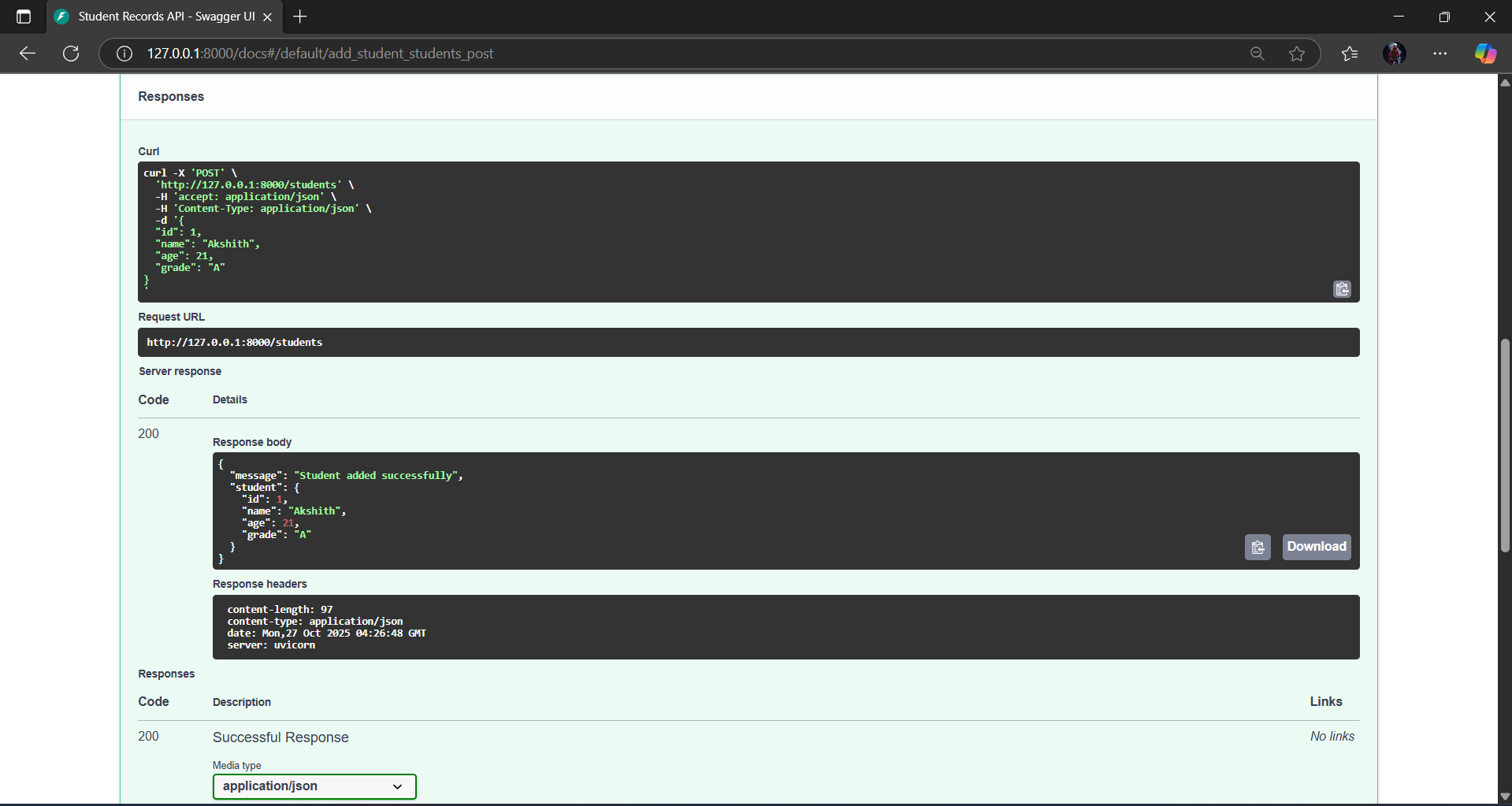
**Code:**

****

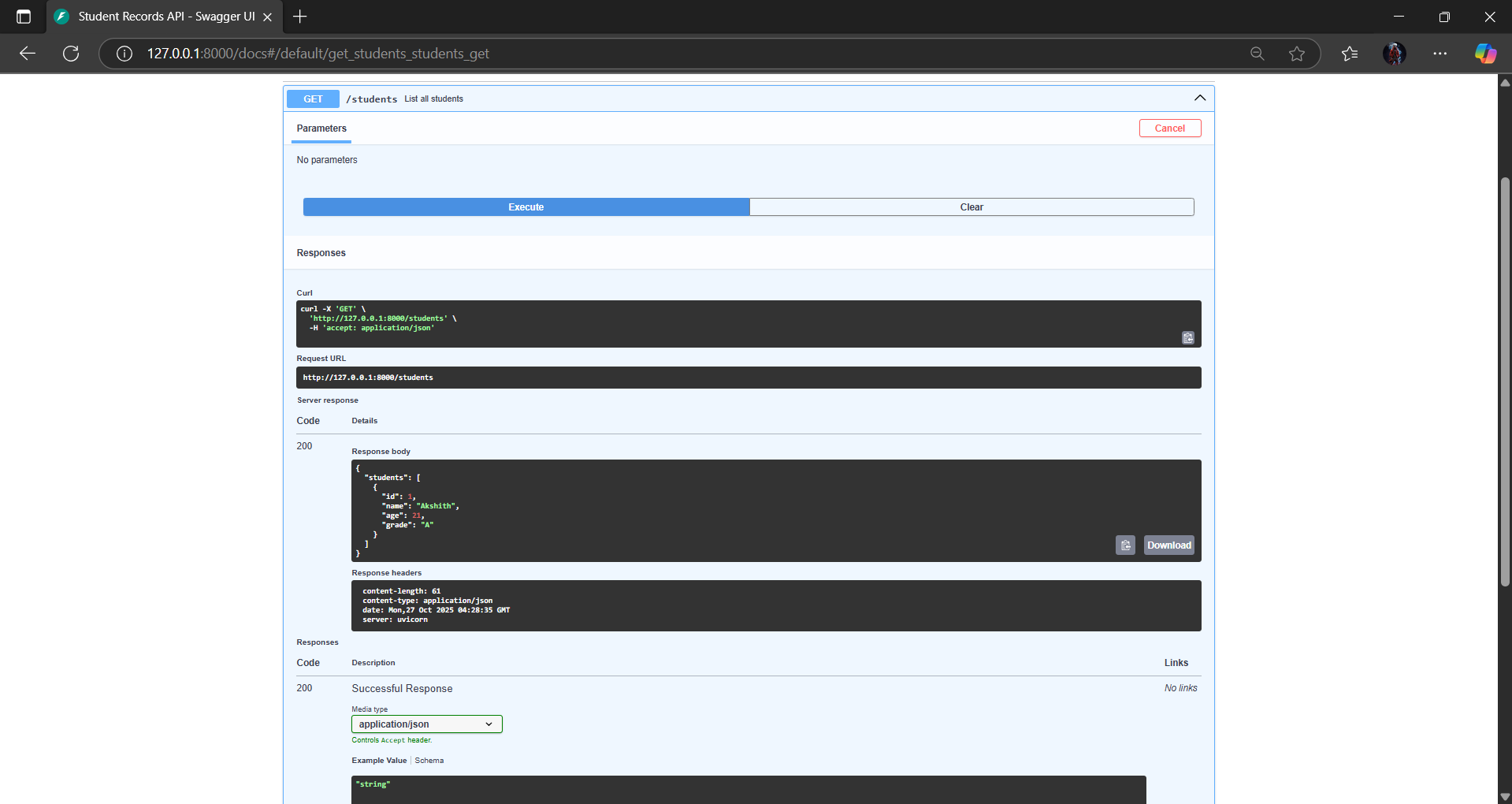
**API Operations/Output:**

**1)Add Student (POST /students)**

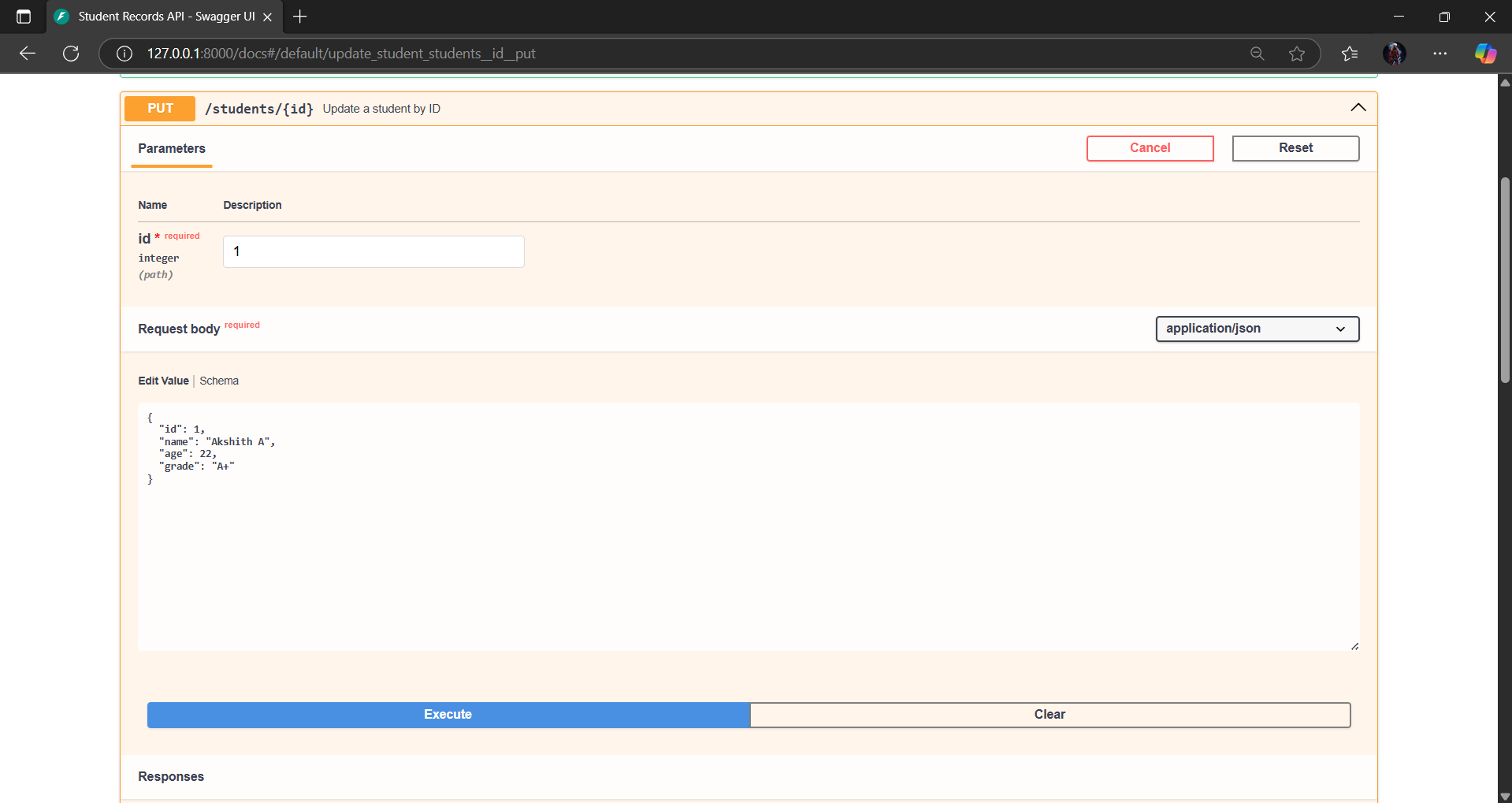
****

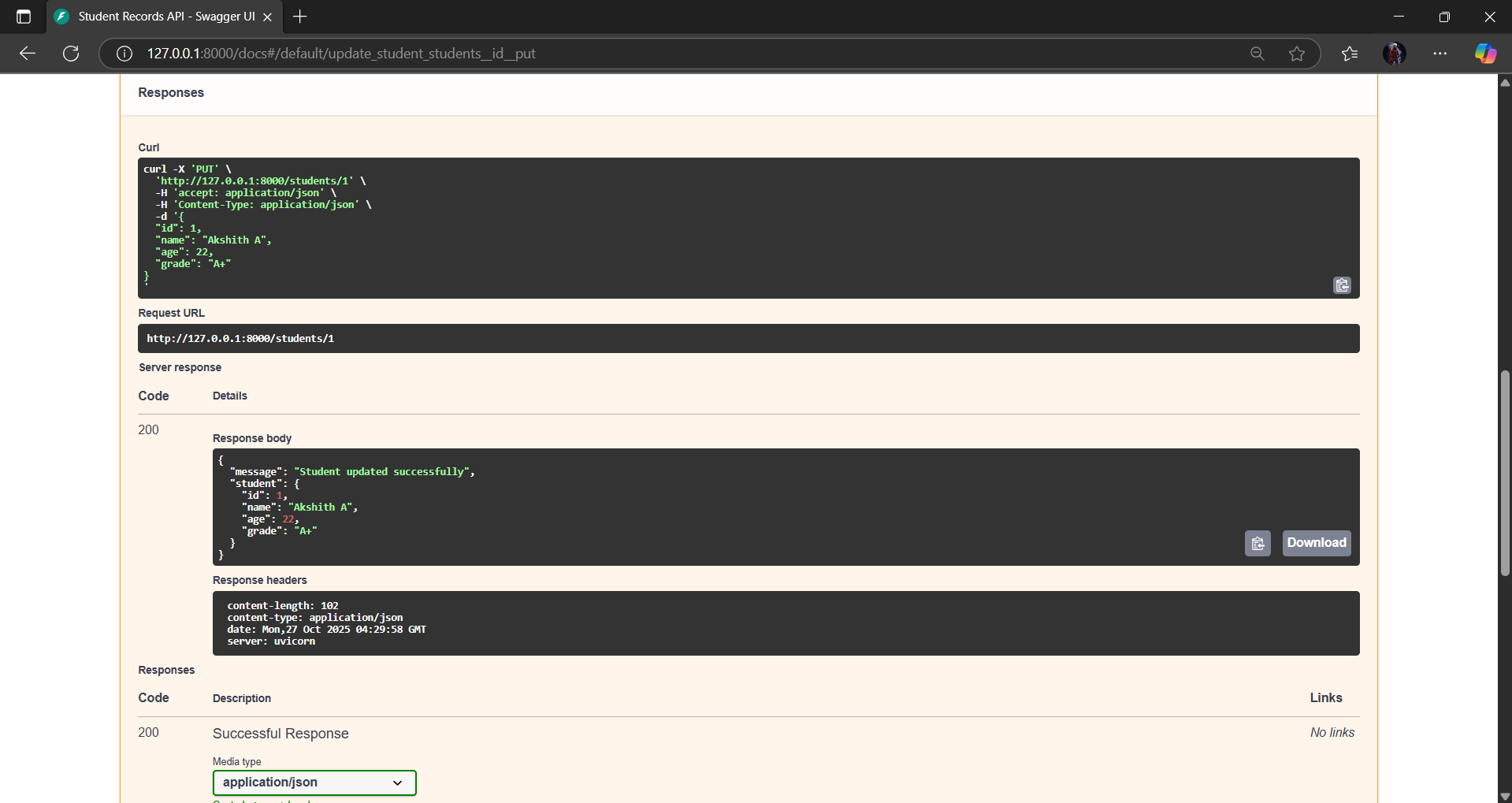
****

**2)** **Get All Students (GET /students)**

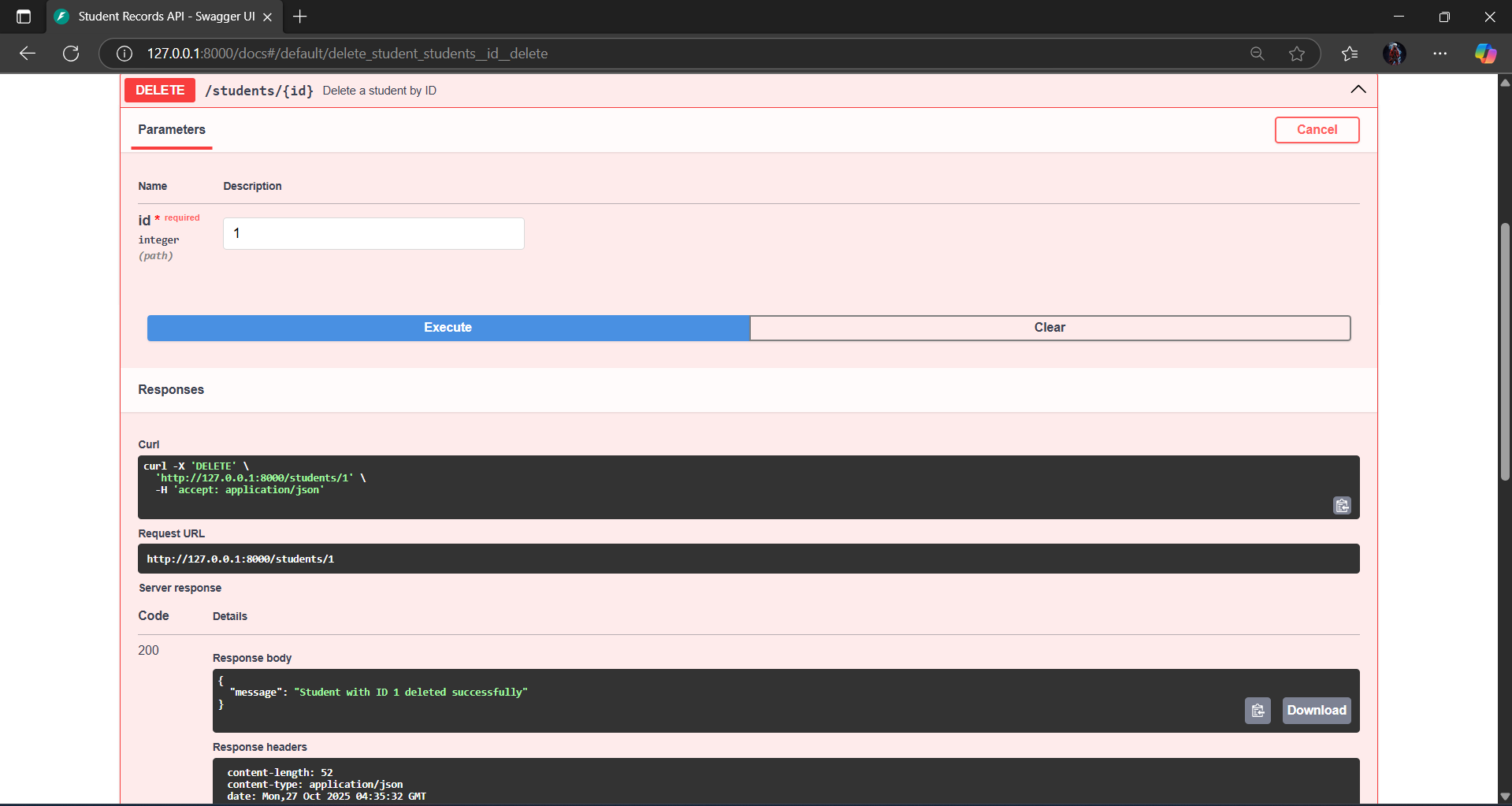
****

**3)** **Update Student (PUT /students/1)**

****

****

**4)** **Delete Student (DELETE /students/1)**

****

**Observation**

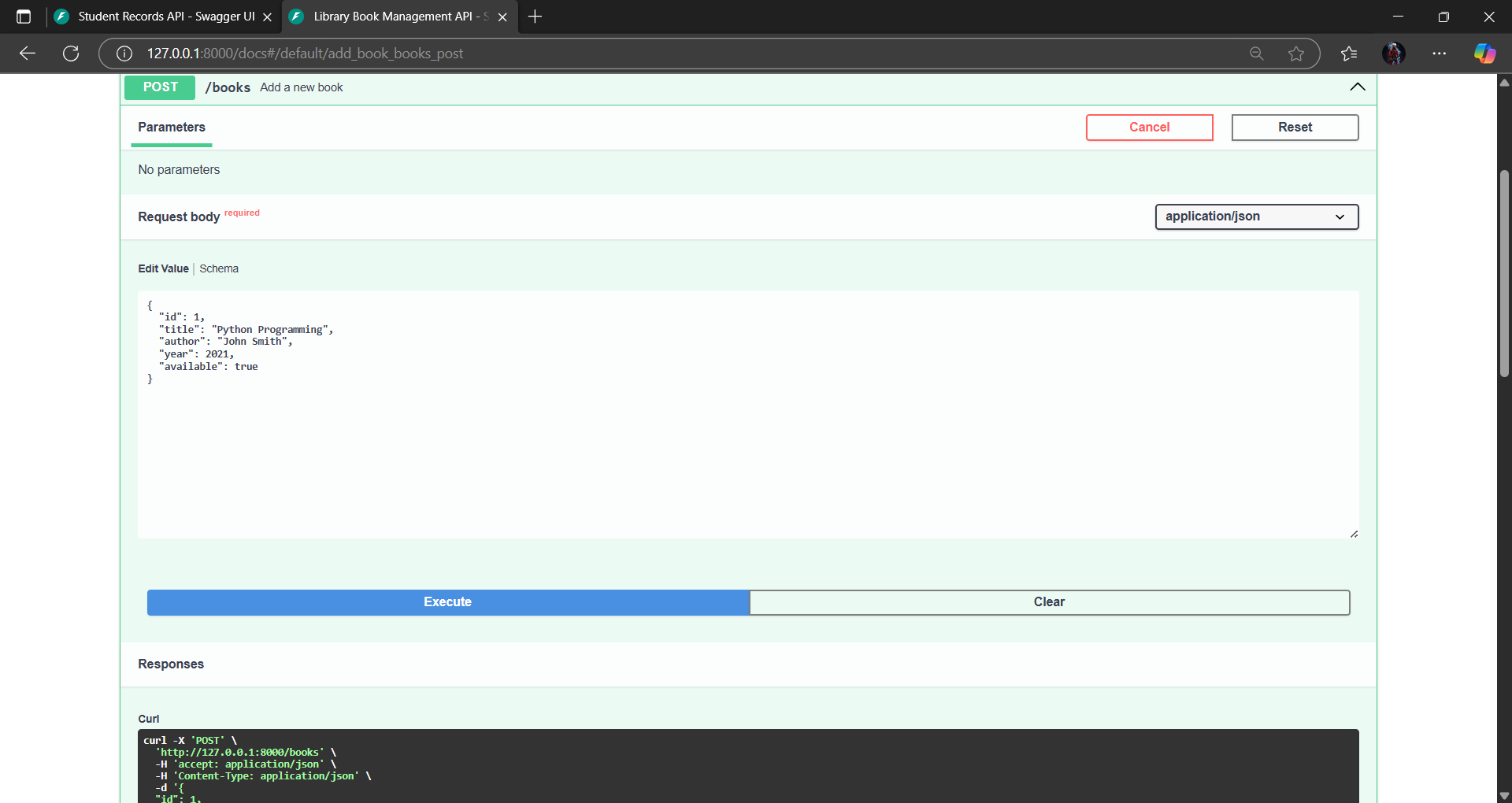
* The RESTful API follows proper CRUD principles.
* Data is stored temporarily in memory using a Python dictionary.
* FastAPI automatically generates API documentation (/docs).
* JSON is used as the standard response format.
* Each endpoint handles errors (e.g., duplicate ID, missing record) with clear messages.

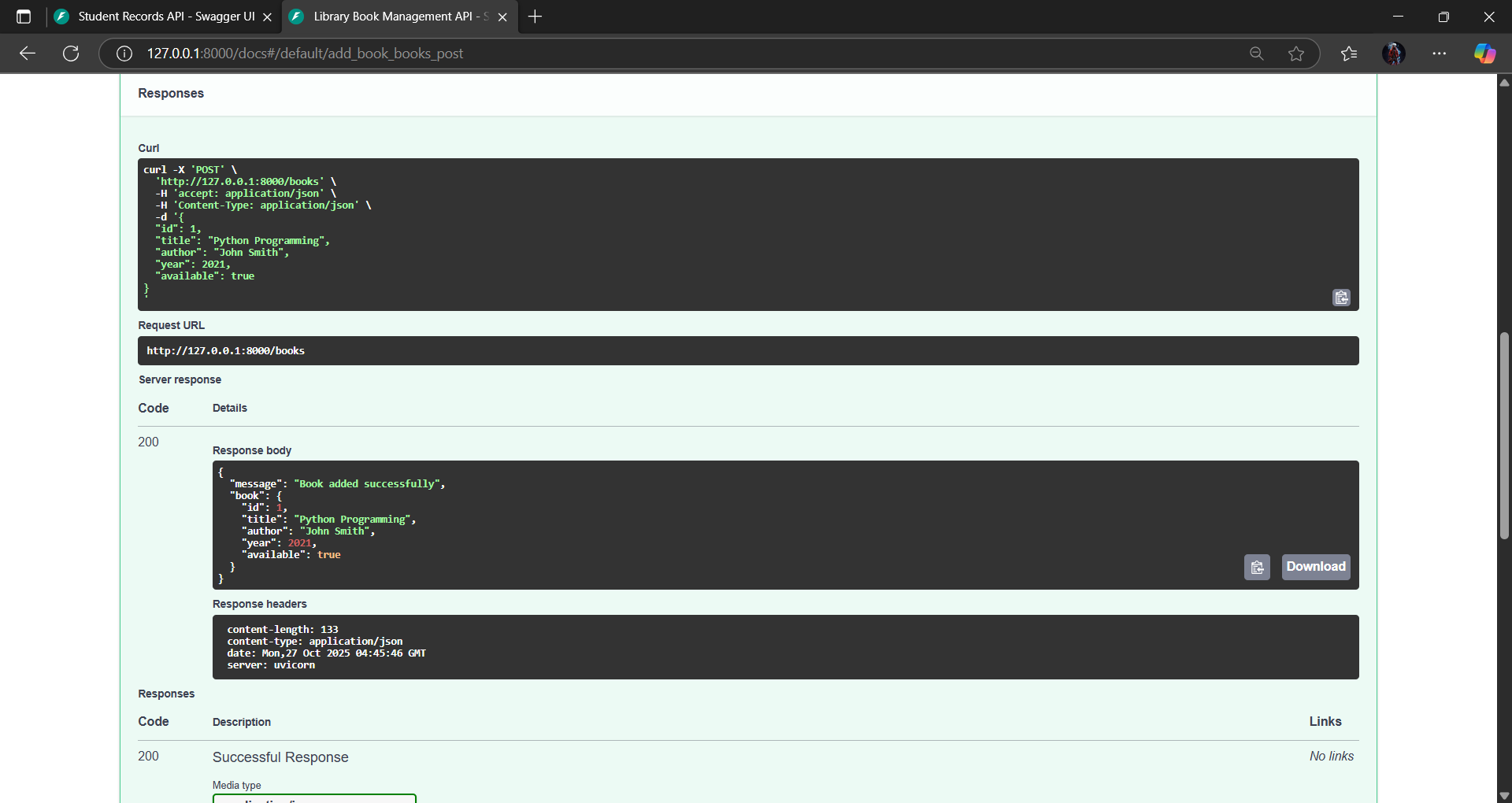
**Task 2 – Library Book Management API**

**Prompt:** For **Task 2 – Library Book Management API**, I need to develop a RESTful API to manage library books. The API should include several endpoints: GET /books to retrieve all books, POST /books to add a new book, GET /books/{id} to get the details of a specific book, PATCH /books/{id} to update partial details such as book availability, and DELETE /books/{id} to remove a book from the collection. The implementation must include proper **error handling** for invalid requests or missing data. The expected outcome is a **fully functional API** that supports **CRUD operations** along with **partial updates** for specific book attributes.

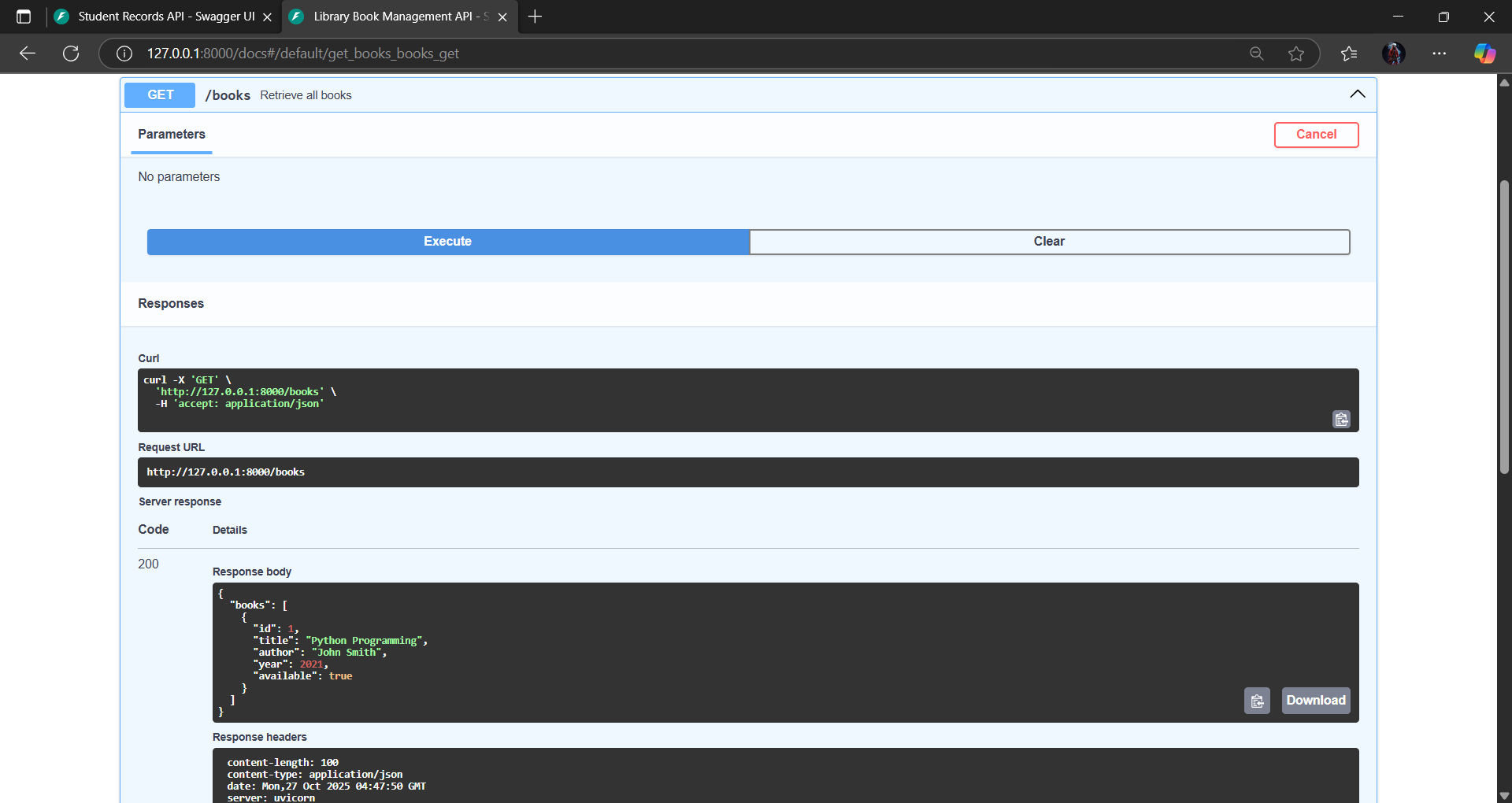
**API Operations/Output:**

1. **Add Book (POST /books)**

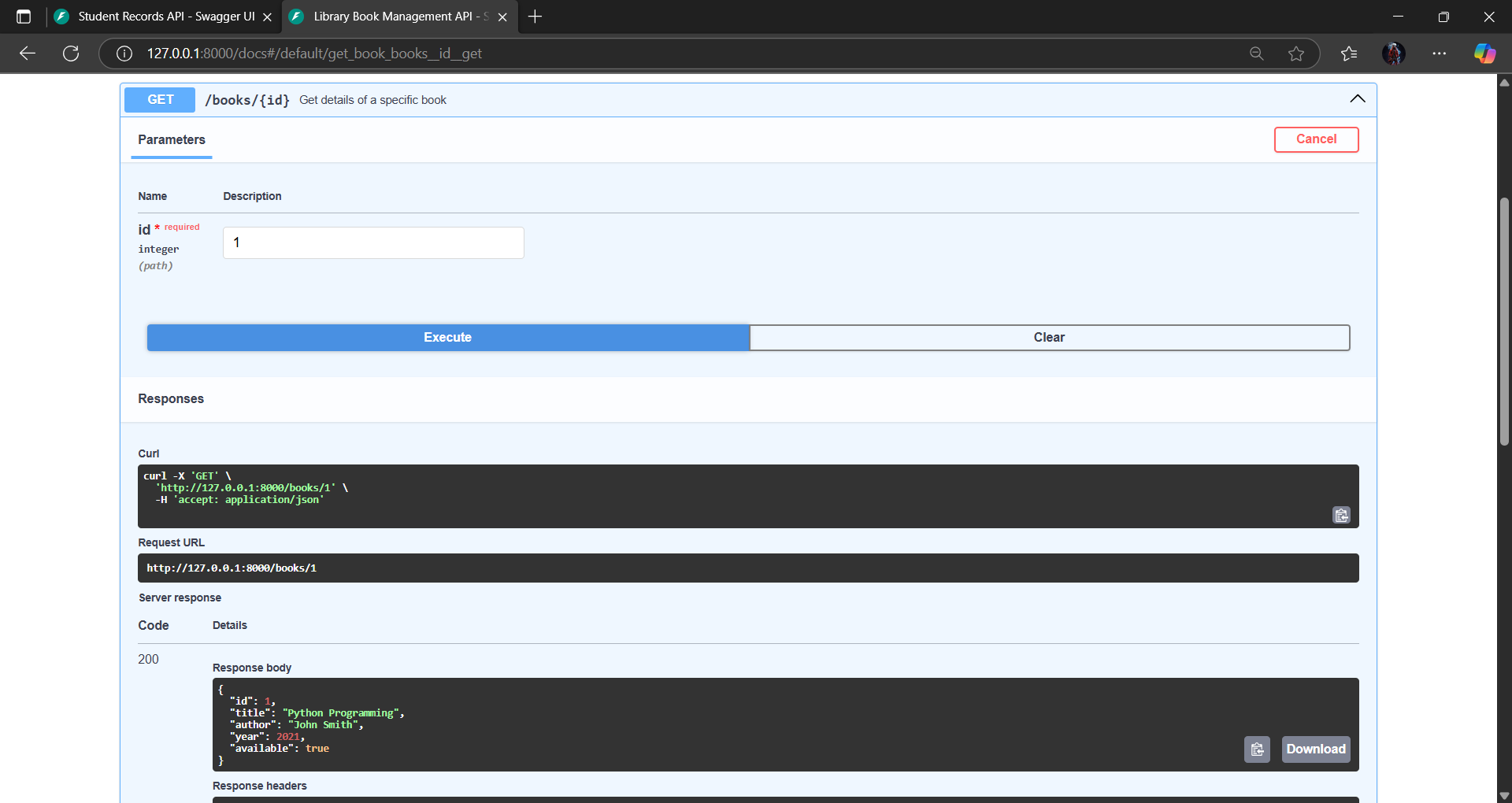
****

****

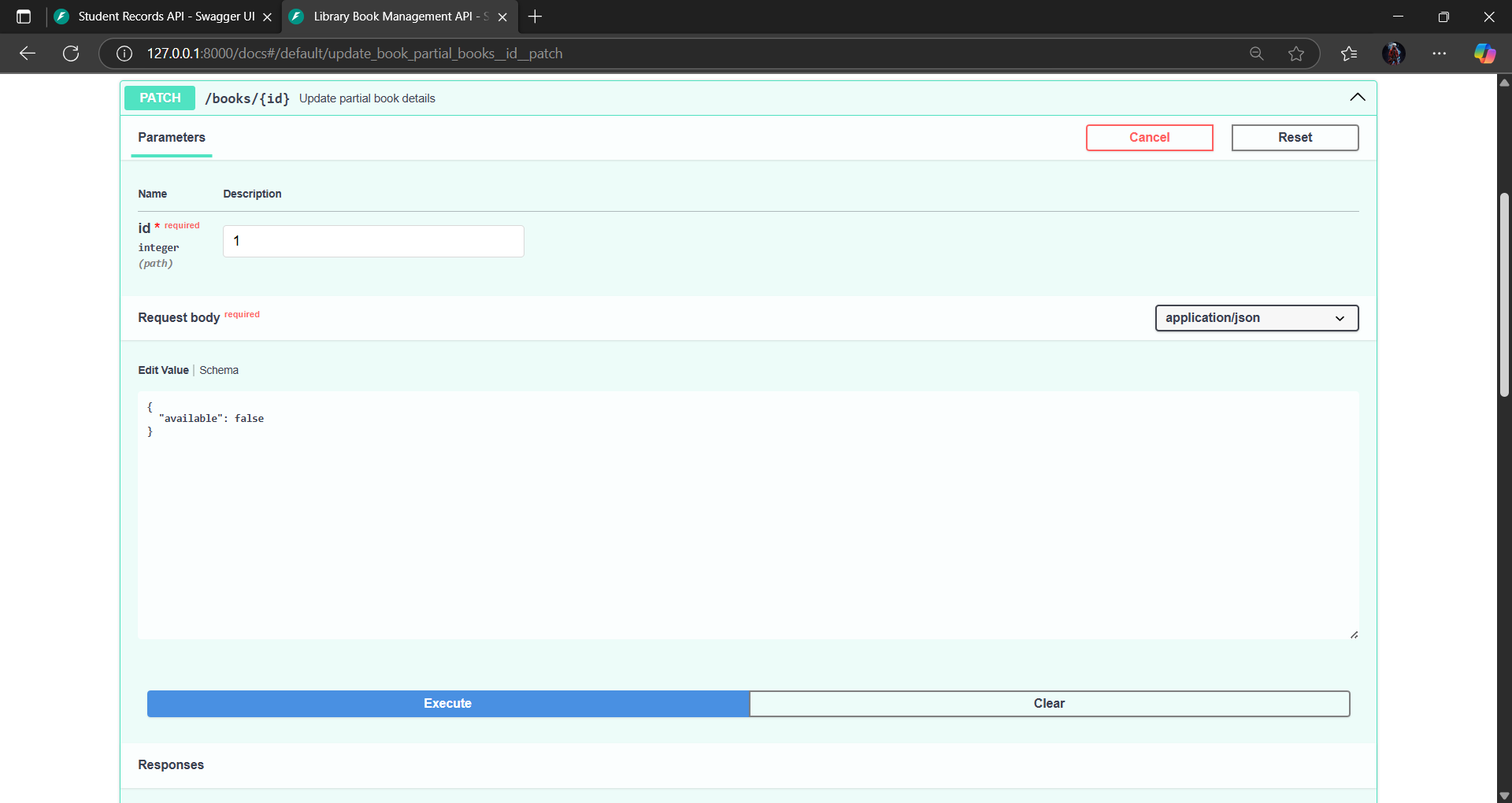
**2)Retrieve All Books (GET /books)**

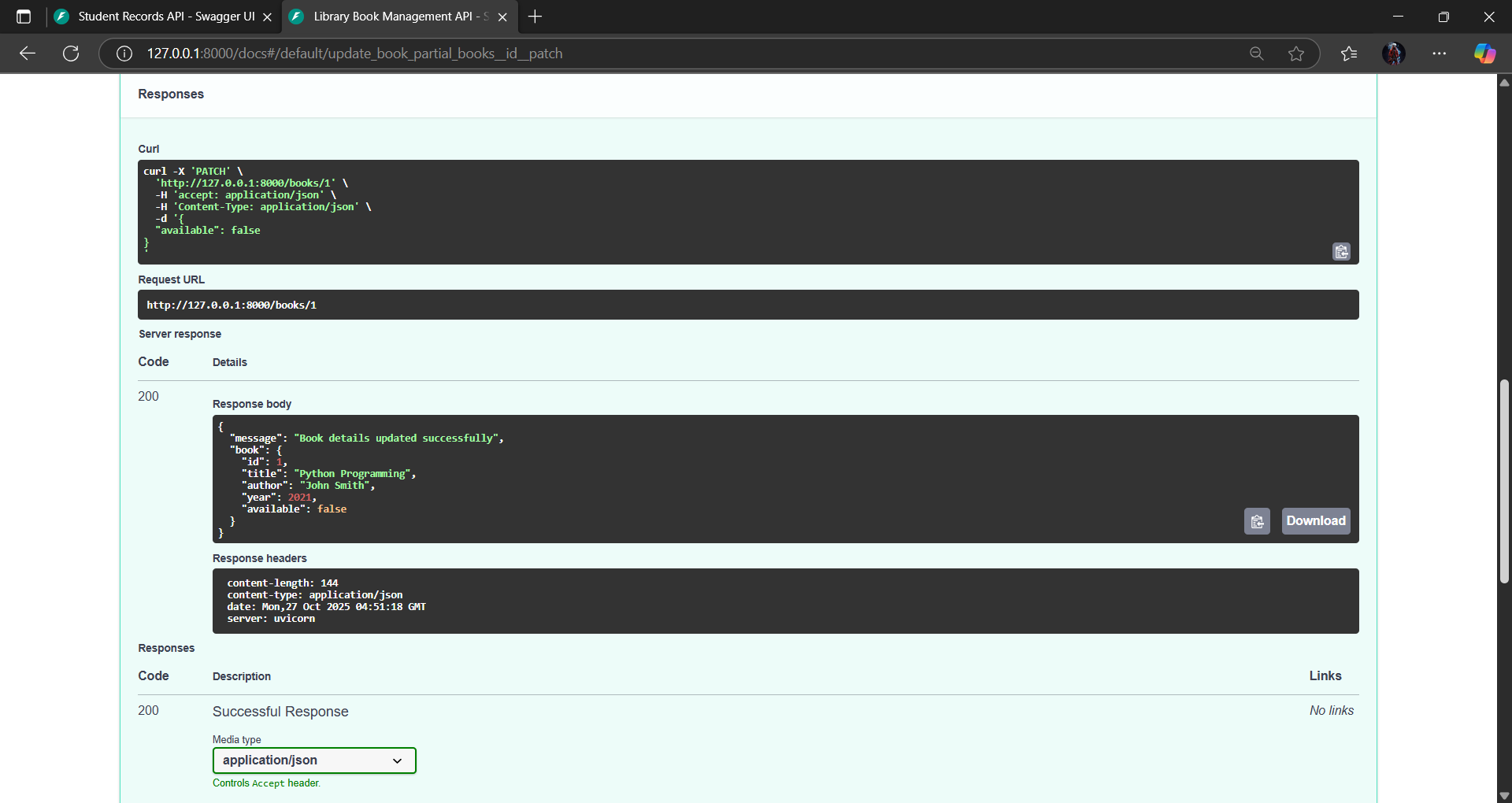
****

**3) Get Book by ID (GET /books/1)**

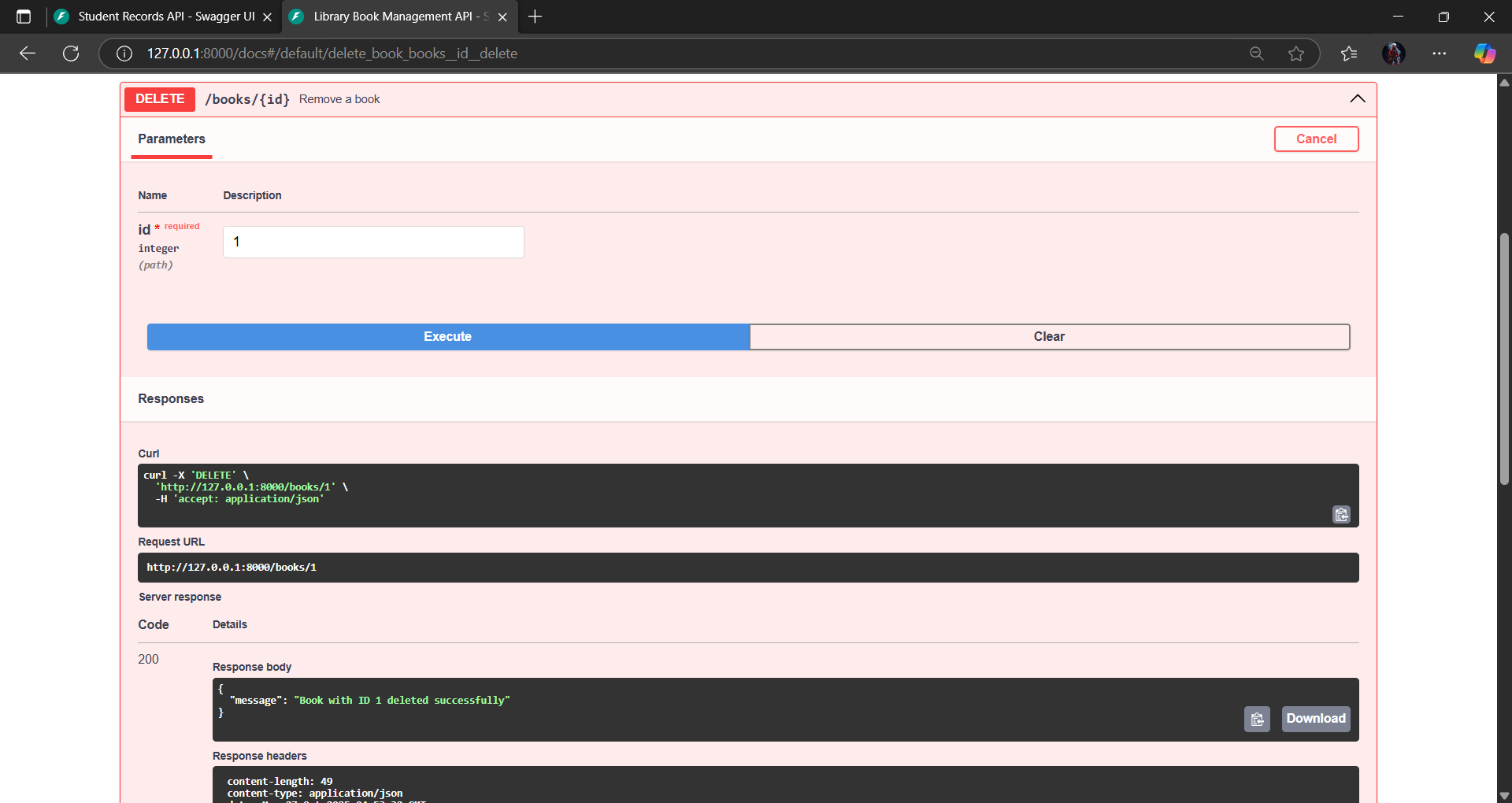
****

**4)** **Update Partial Details (PATCH /books/1)**

****

****

**5)** **Delete a Book (DELETE /books/1)**

****

### ****Observation****

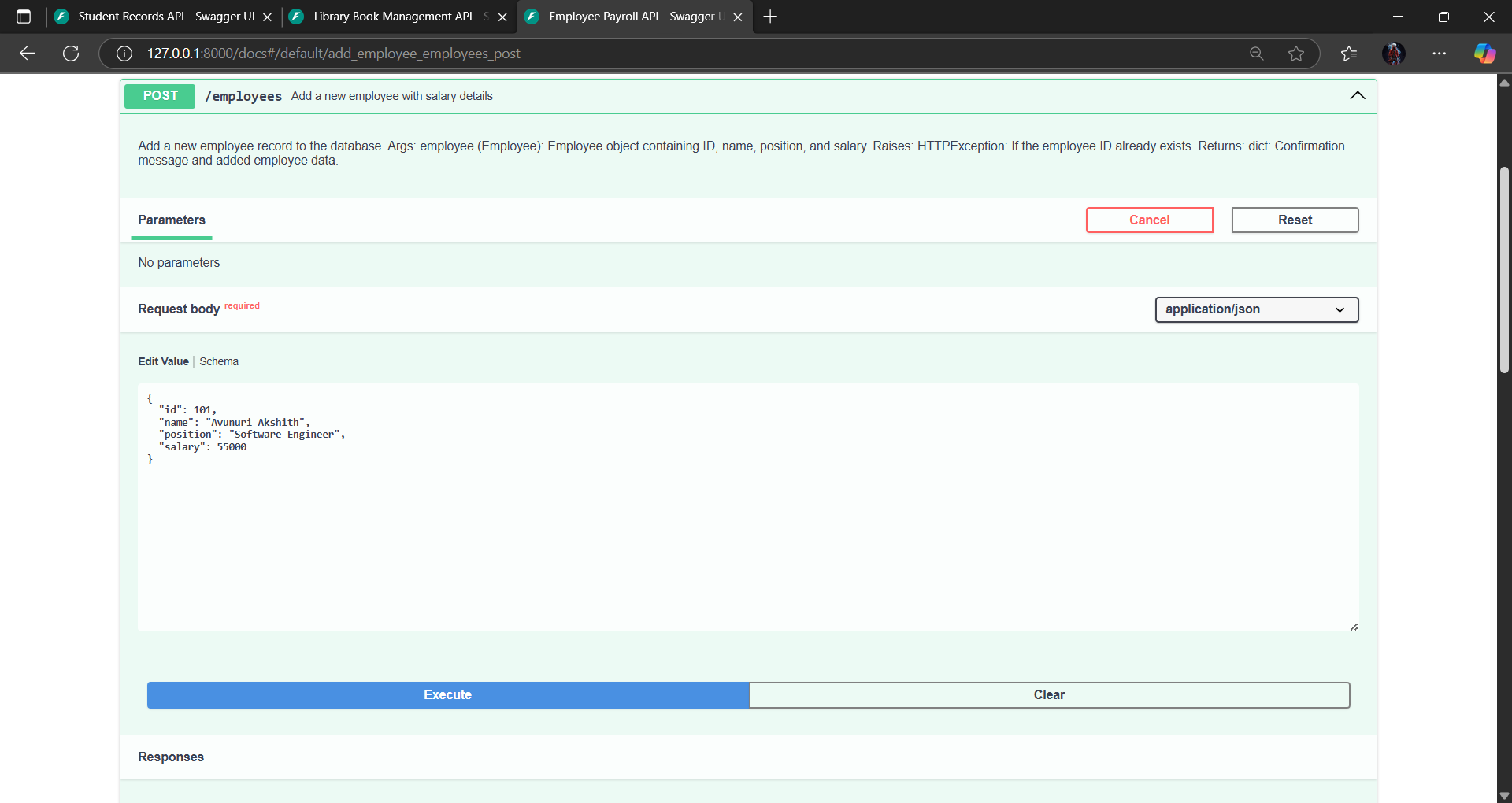
* The API successfully performs **CRUD operations** and supports **partial updates** using the PATCH method.
* Data is temporarily stored in an **in-memory dictionary**, simulating a database.
* Each endpoint returns structured **JSON responses**.
* Proper **error handling** ensures users receive clear messages when invalid IDs or duplicate entries are provided.
* The **auto-generated FastAPI documentation** (/docs) makes it easy to test and understand each endpoint

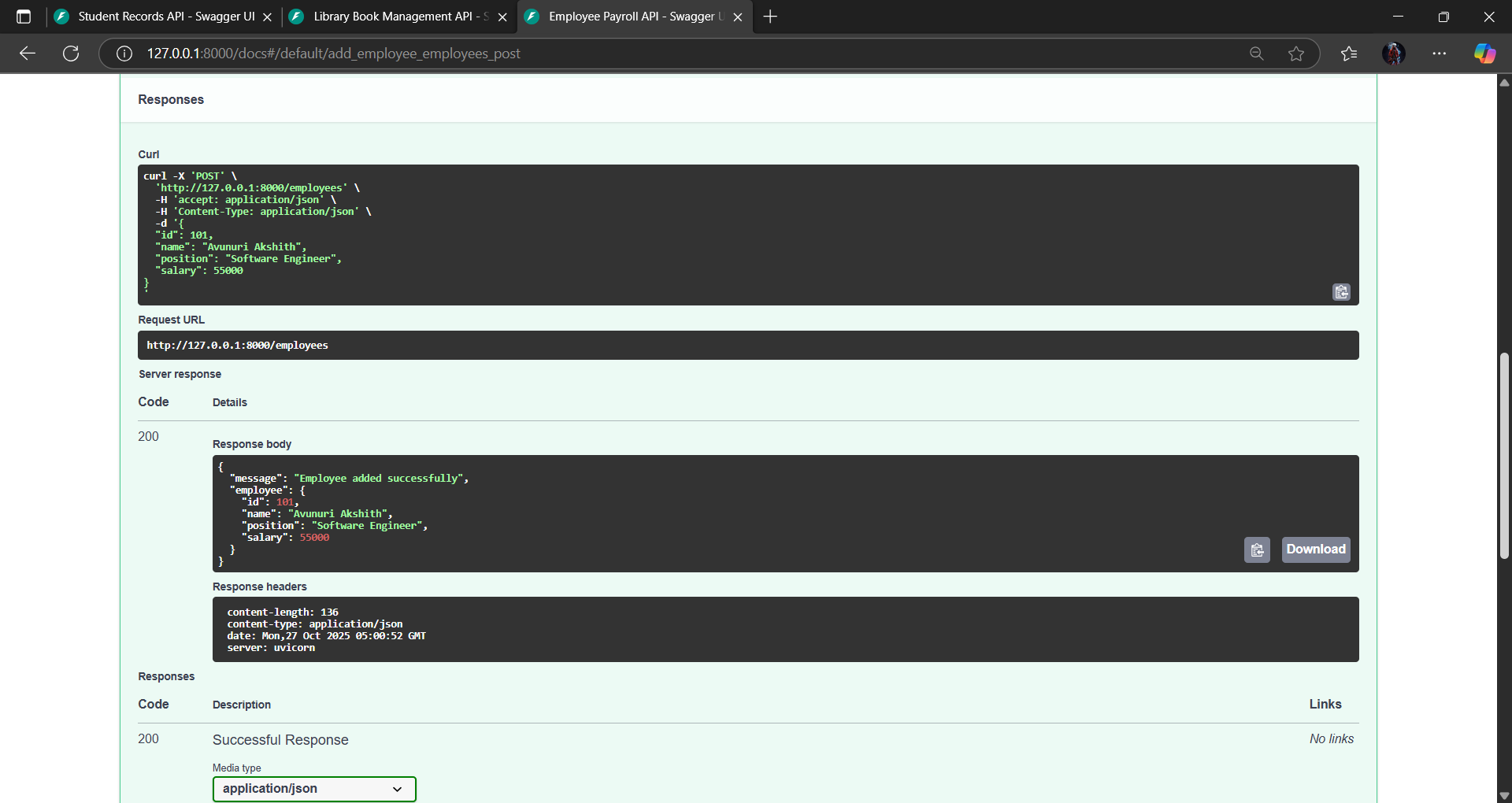
**Task 3 – Employee Payroll API**

**Prompt:** For **Task 3 – Employee Payroll API**, I need to build a RESTful API that manages employees and their salary information. The API should have endpoints to list all employees, add new employees with salary details, update an employee’s salary, and delete an employee record. Specifically, the endpoints are GET /employees to list employees, POST /employees to add a new employee, PUT /employees/{id}/salary to update an employee’s salary, and DELETE /employees/{id} to remove an employee from the system. AI assistance should be used to suggest the data model structure and to include comments and docstrings for every endpoint. The final output should be a well-documented API that supports salary management and demonstrates clear endpoint documentation.

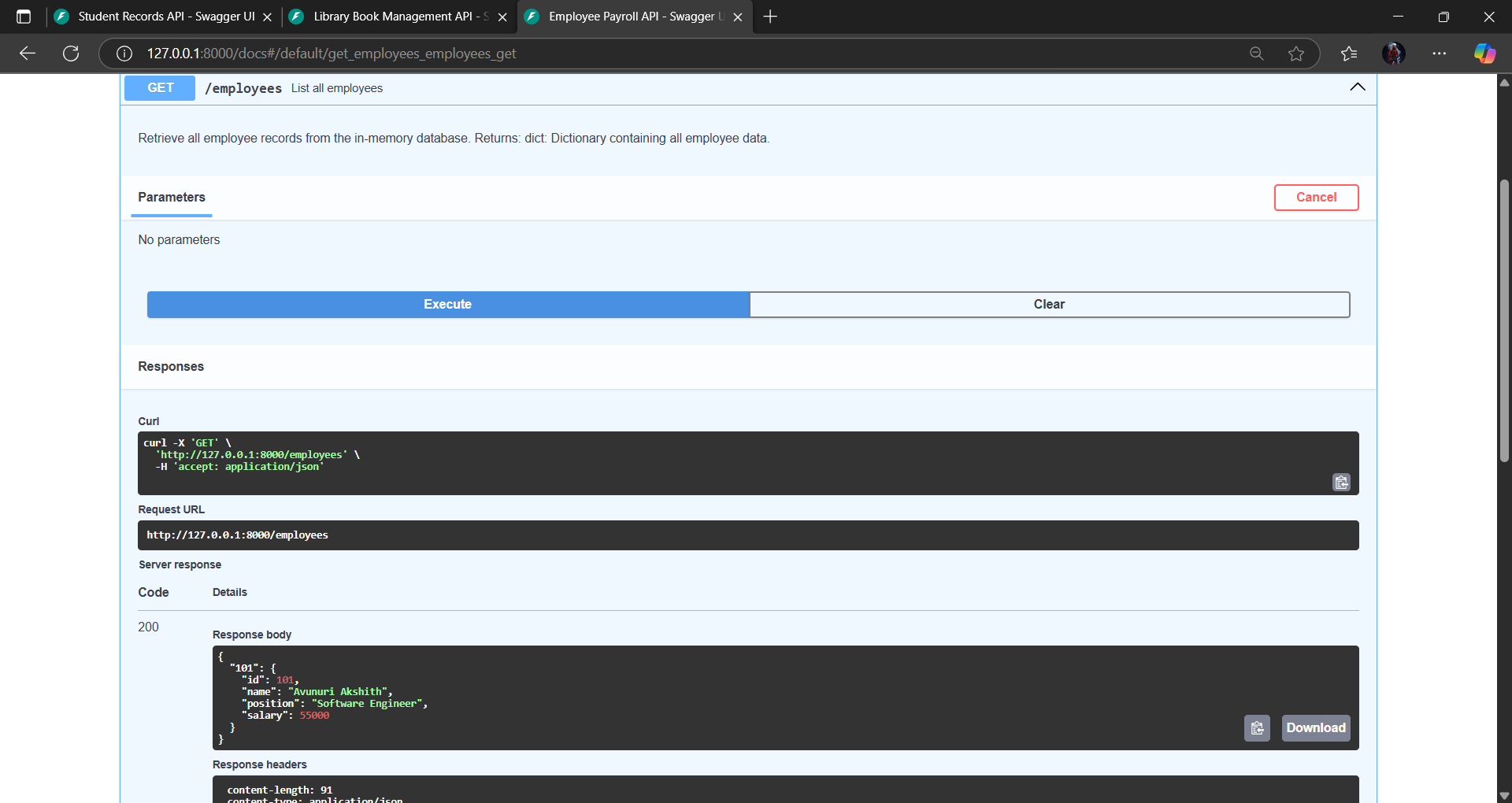
**API Operations/Output:**

1. **Add Employee (POST /employees)**

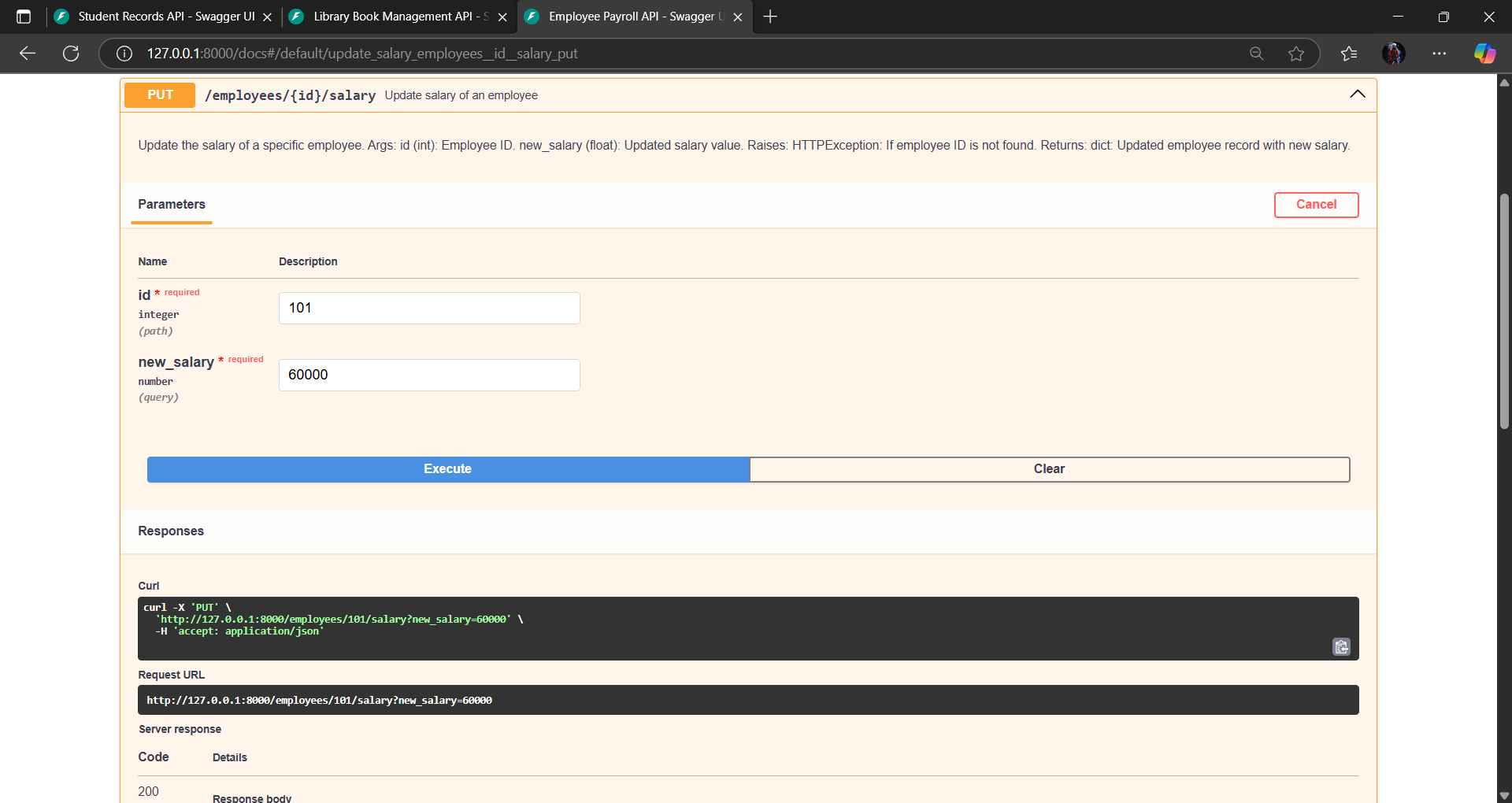
****

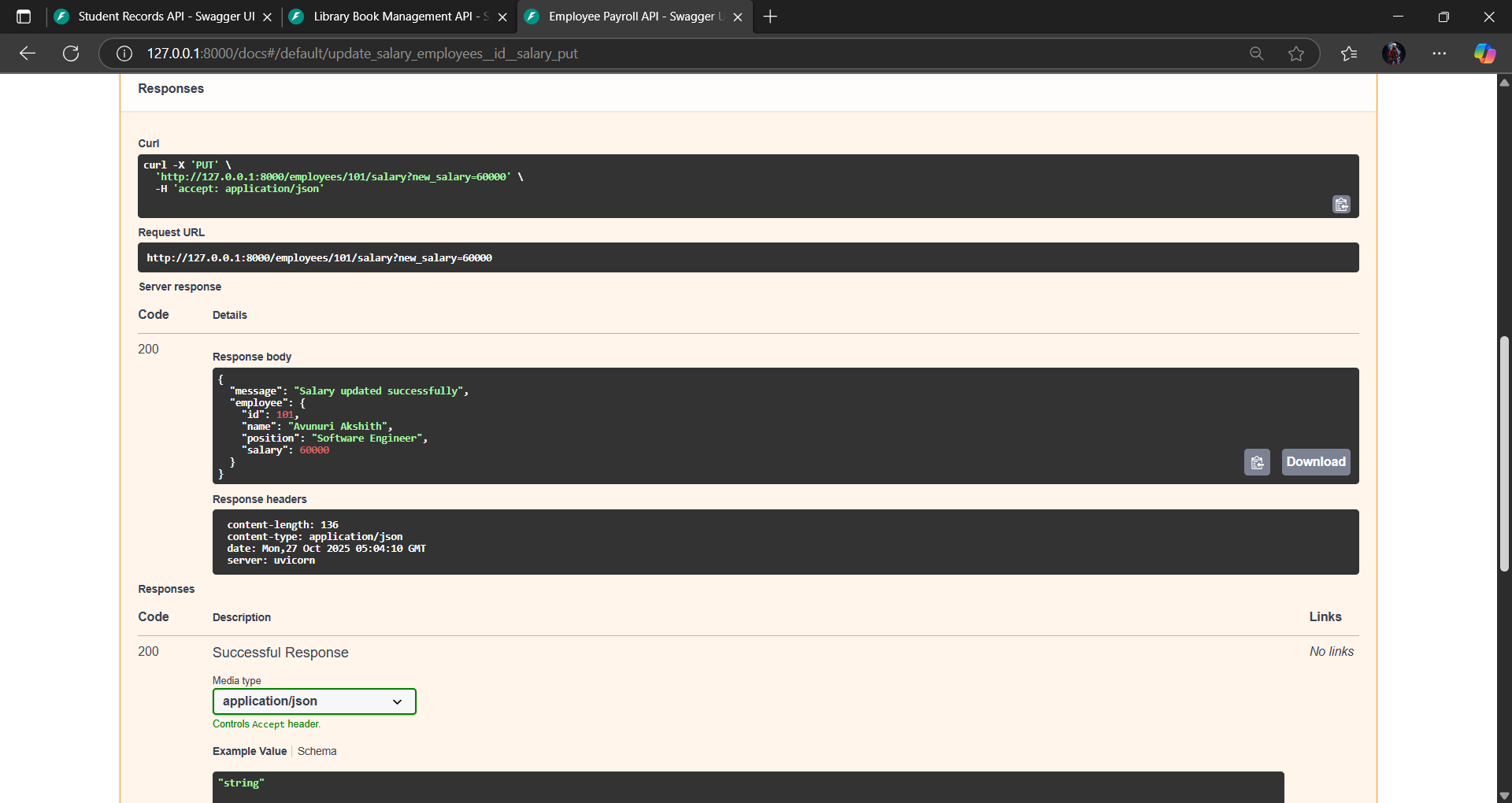
****

1. **Get All Employees (GET /employees)**

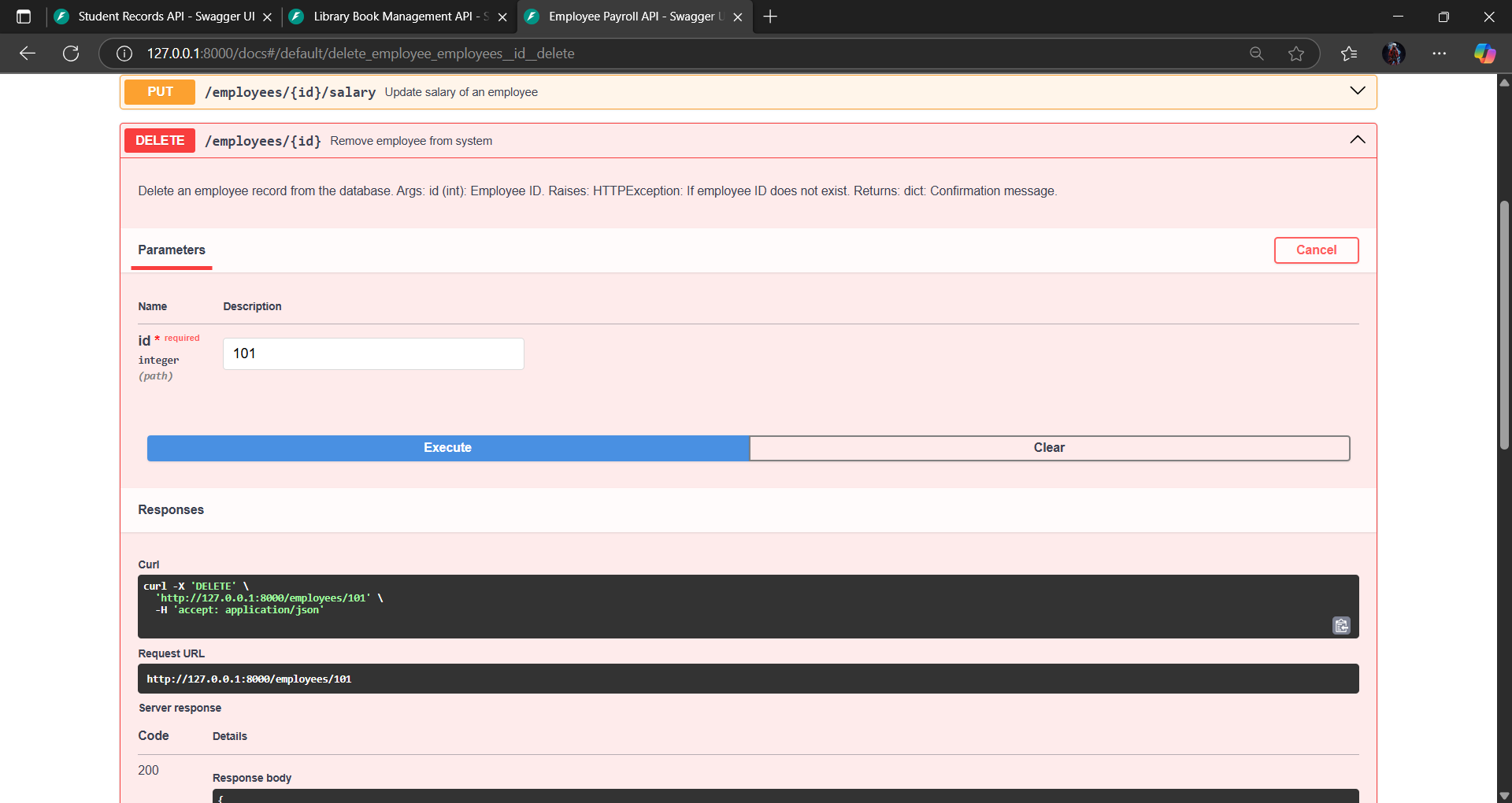
****

**3)Update Salary (PUT /employees/101/salary?new\_salary=60000)**

****

****

1. **Delete Employee (DELETE /employees/101)**

****

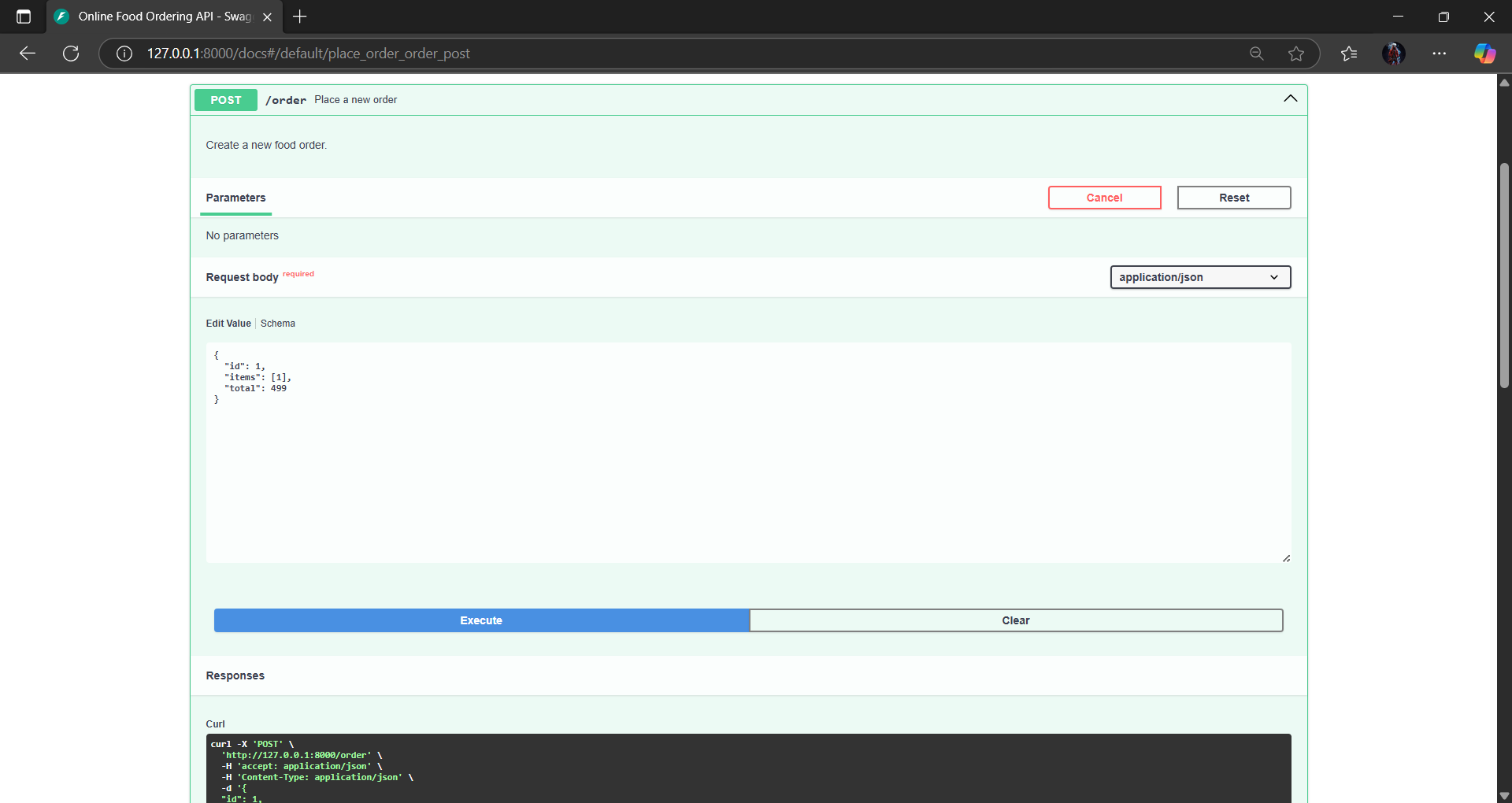
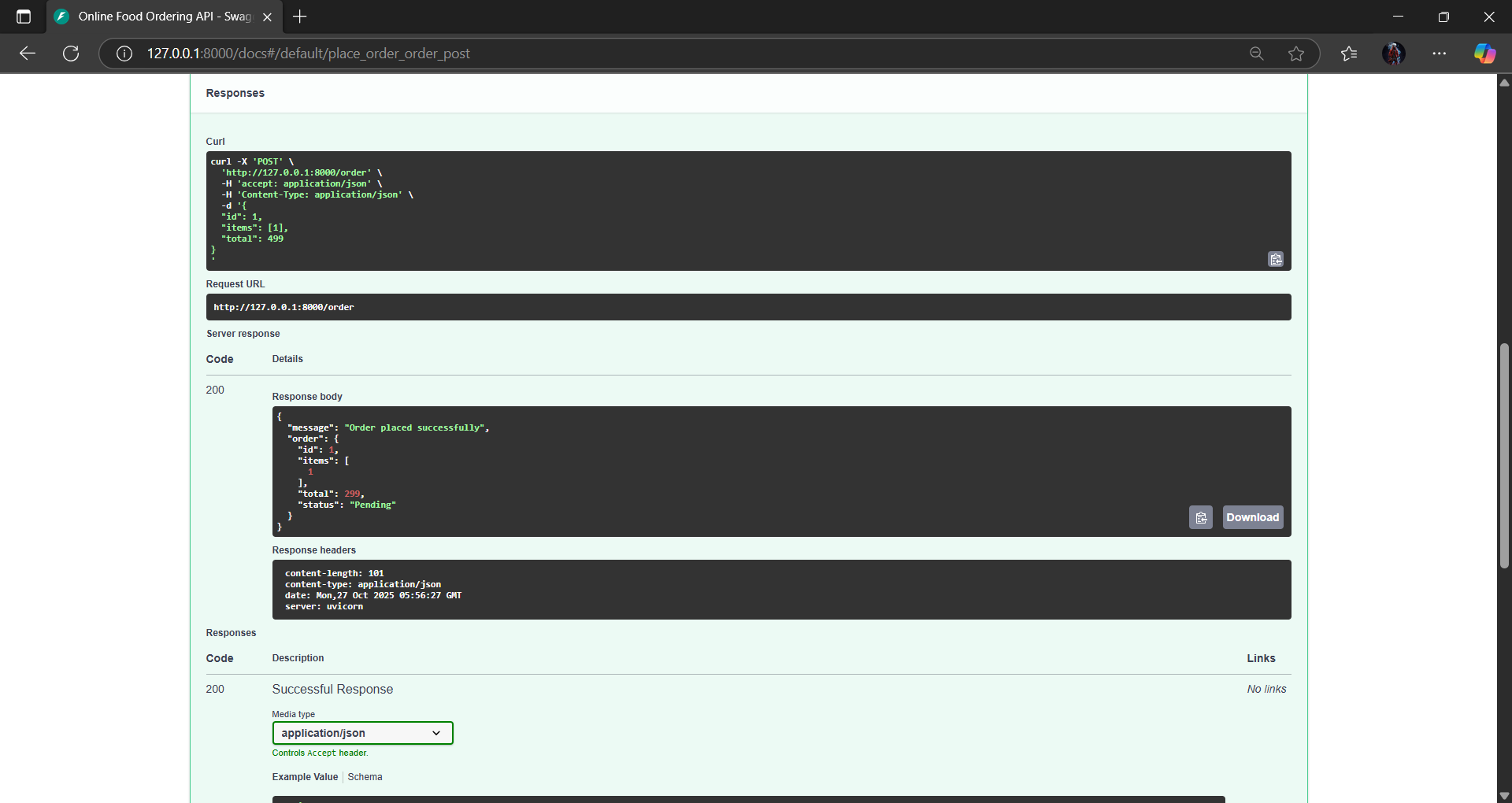
### ****Observation****

* The API performs all **CRUD operations** plus salary updates effectively.
* Data is managed in an **in-memory dictionary**, which makes testing fast and simple.
* Each endpoint is thoroughly **documented with comments and docstrings**, improving readability and maintainability.
* The API includes **error handling** for duplicate IDs and missing employees.
* FastAPI automatically generates **interactive documentation** at /docs, allowing users to test endpoints easily.
* Overall, this API provides a clean and well-structured foundation for a full payroll management system.

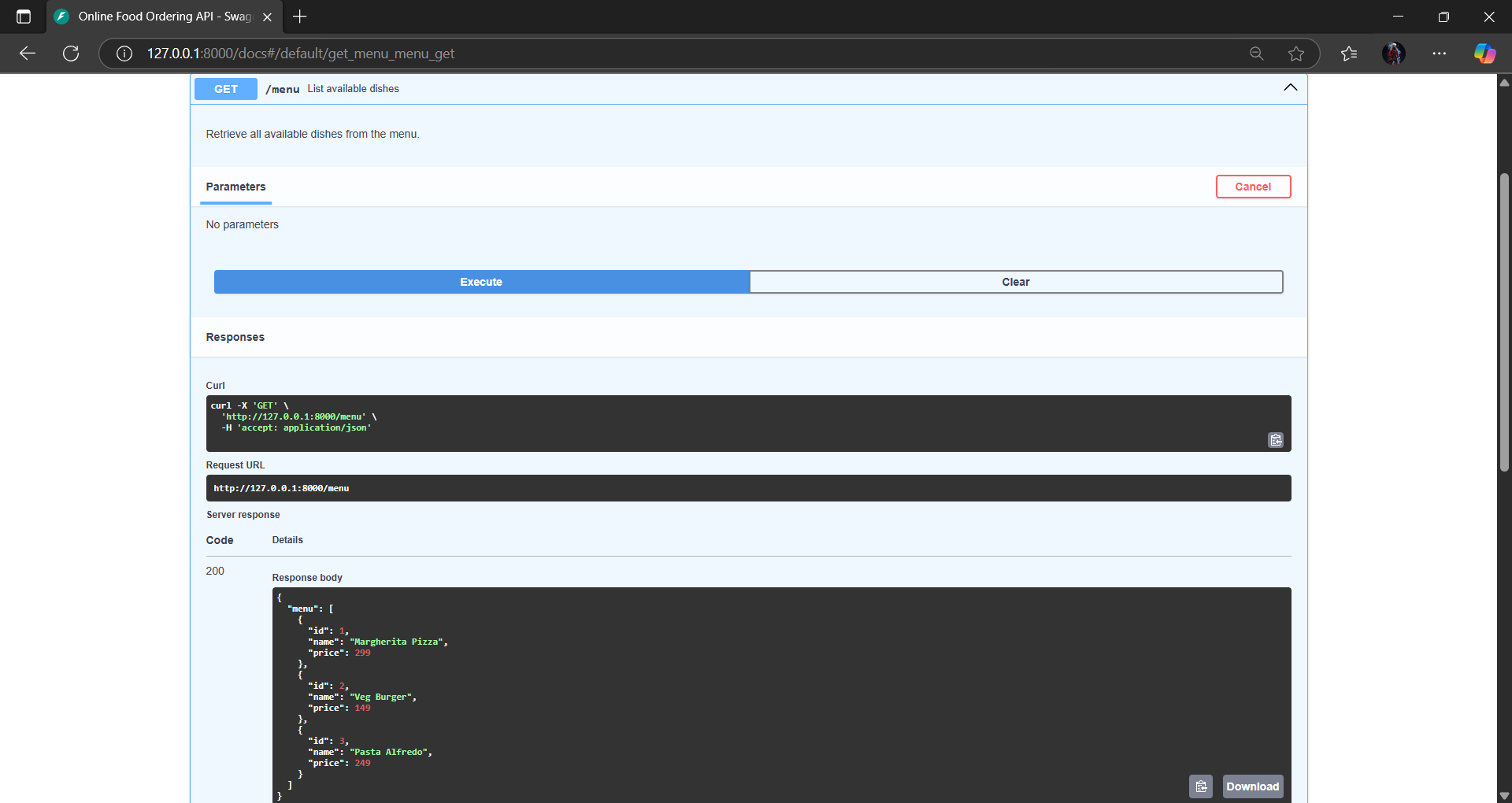
**Task 4 – Real-Time Application: Online Food Ordering API Scenario:**

**Prompt:** For **Task 4 – Online Food Ordering API**, I need to design a simple RESTful backend for an online food ordering system. The API should include endpoints to view the menu, place a new order, check the status of an order, update an existing order (such as changing items), and cancel an order. Specifically, the endpoints are GET /menu, POST /order, GET /order/{id}, PUT /order/{id}, and DELETE /order/{id}. The API should return JSON responses and include error handling for invalid or missing order IDs. Additionally, the AI should generate the initial REST API code, suggest potential improvements such as authentication or pagination, and provide test cases to validate functionality. The expected result is a fully working backend API simulating an online food ordering system.

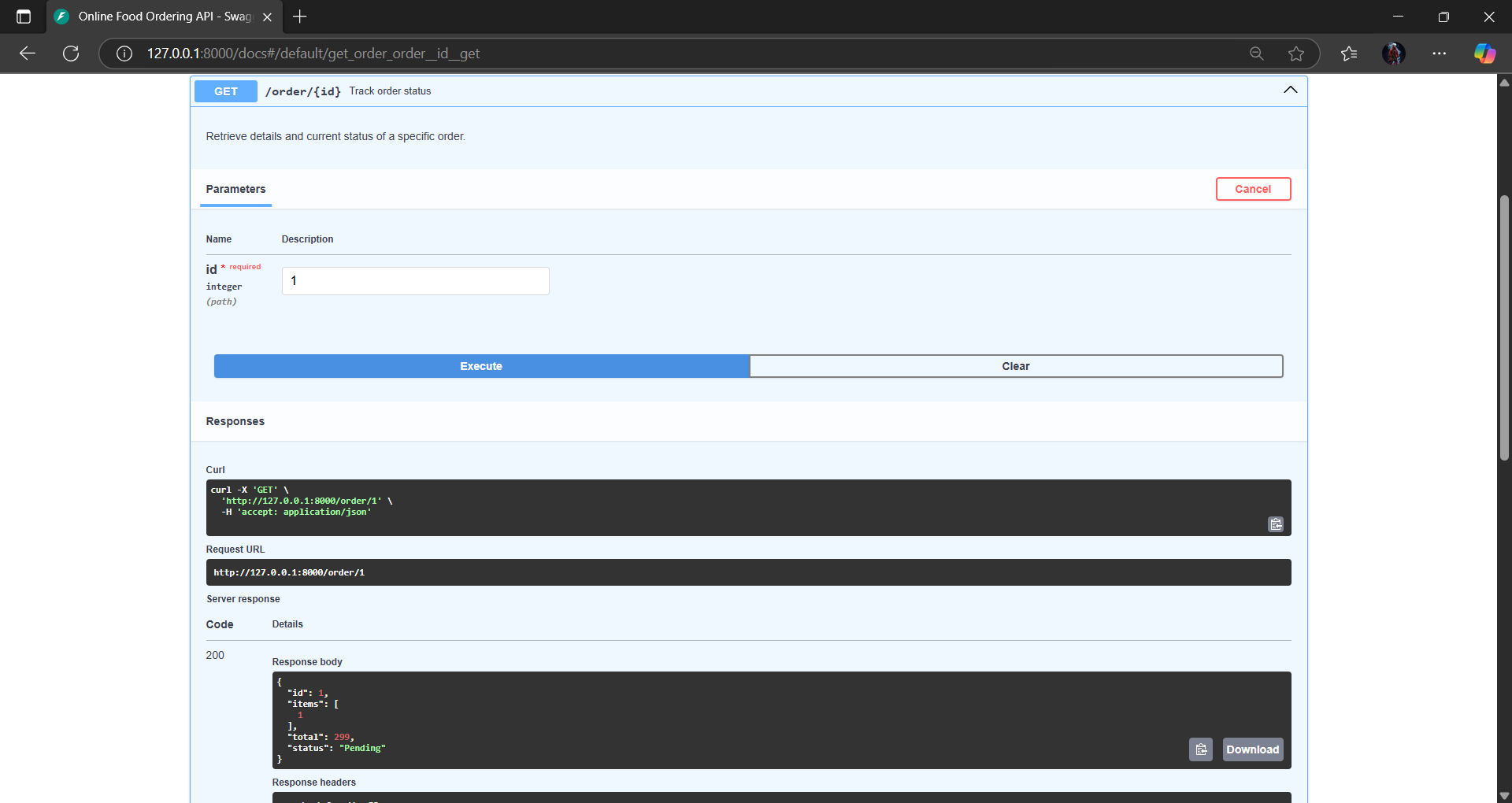
**API Operations/Output:**

1. **Place a new order(POST /Order) **

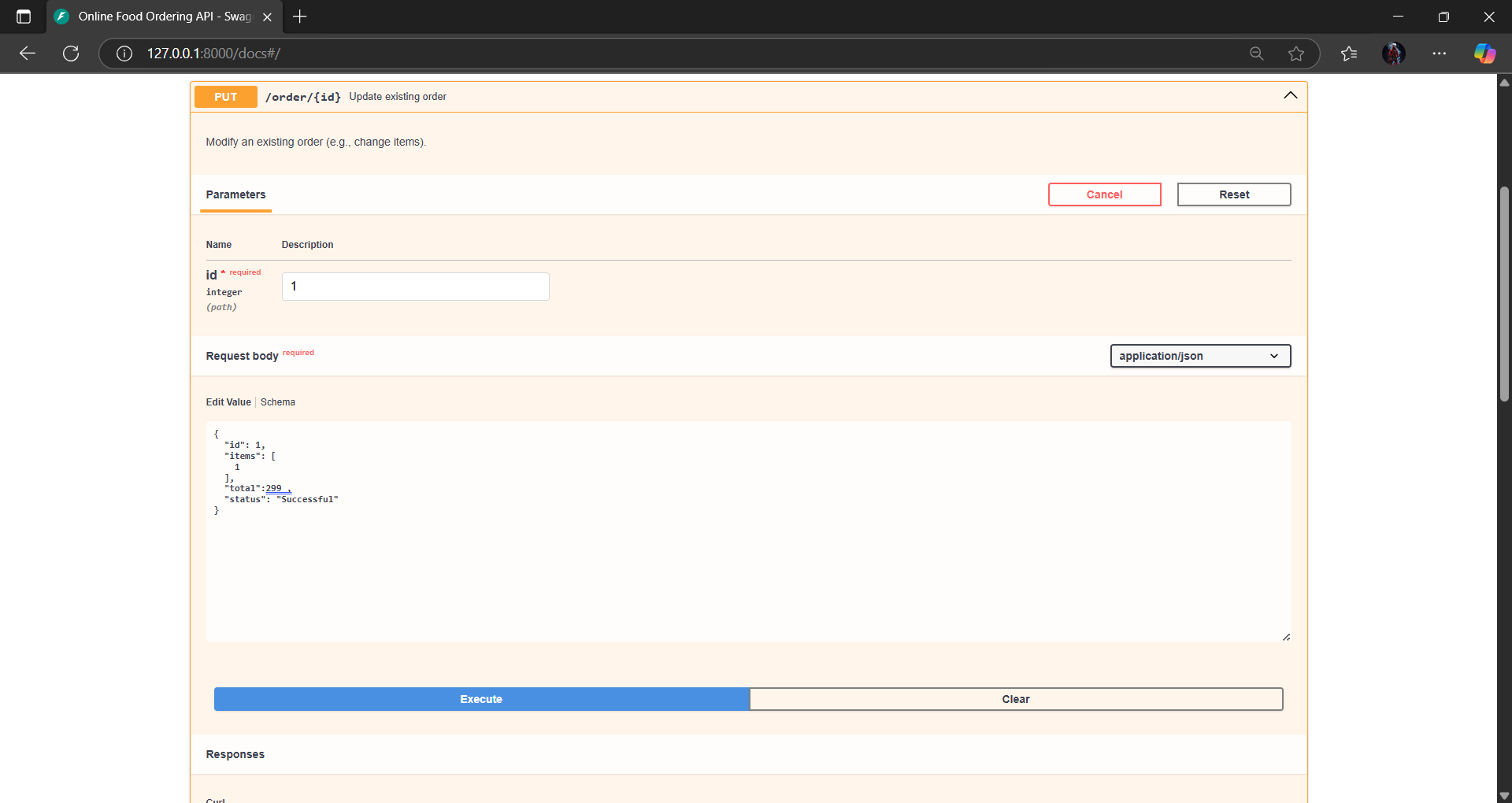
**2)List available dishes(GET/Menu)**

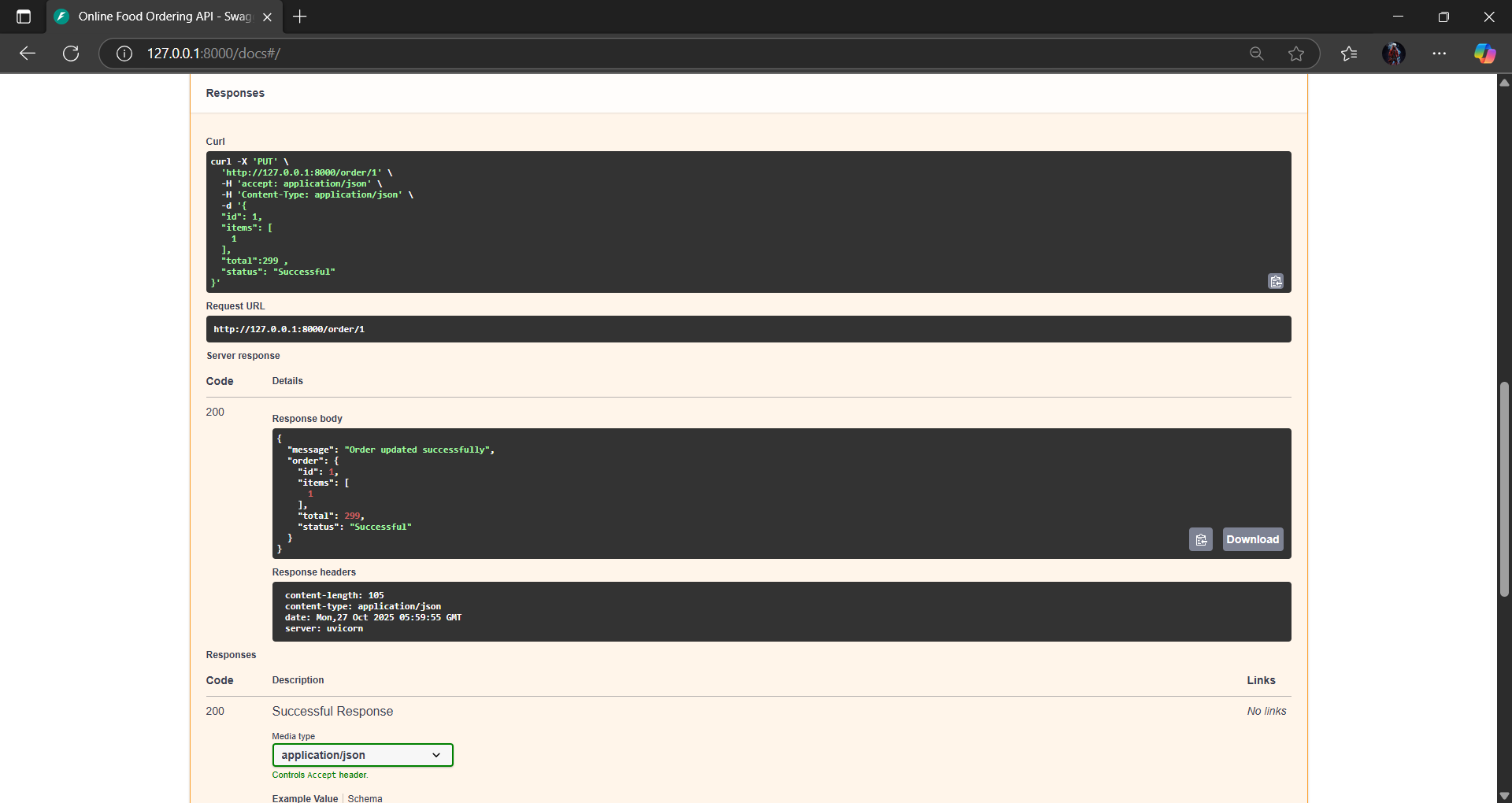
****

**3)Track order status(GET/order)**

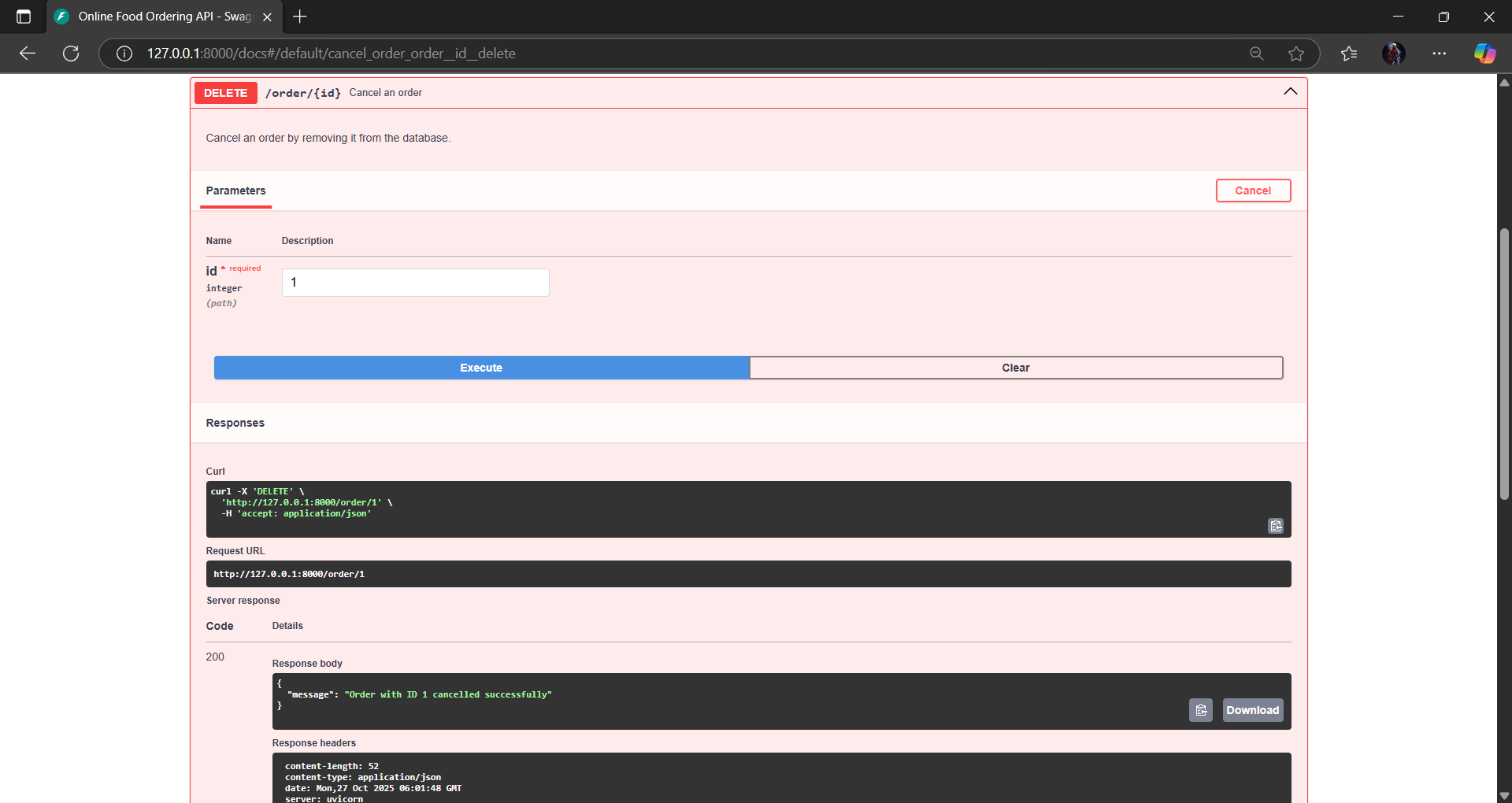
****

**4)Update existing order(put/order)**

****

****

**5)Cancel Order(Delete/Order):**

****

### ****Observation / Analysis:****

* The API successfully implements **CRUD** operations plus **order tracking**.
* Each endpoint responds in **JSON format** and includes **error handling** for missing or invalid IDs.
* Using FastAPI’s **Swagger UI**, endpoints can be tested interactively.
* The API passes all test assertions, confirming correct behavior for order creation, retrieval, and deletion.
* AI-assisted improvements could include:
  + Adding **authentication** for users.
  + Implementing **pagination** for menus.
  + Adding **order timestamps** or **delivery status tracking**.