**Assignment-15**

**Lab 15 – Backend API Development: Creating RESTful Services with AI**

**P.Srinidhi**

**2403a54009**

**Task 1 – Student Records API Task:**

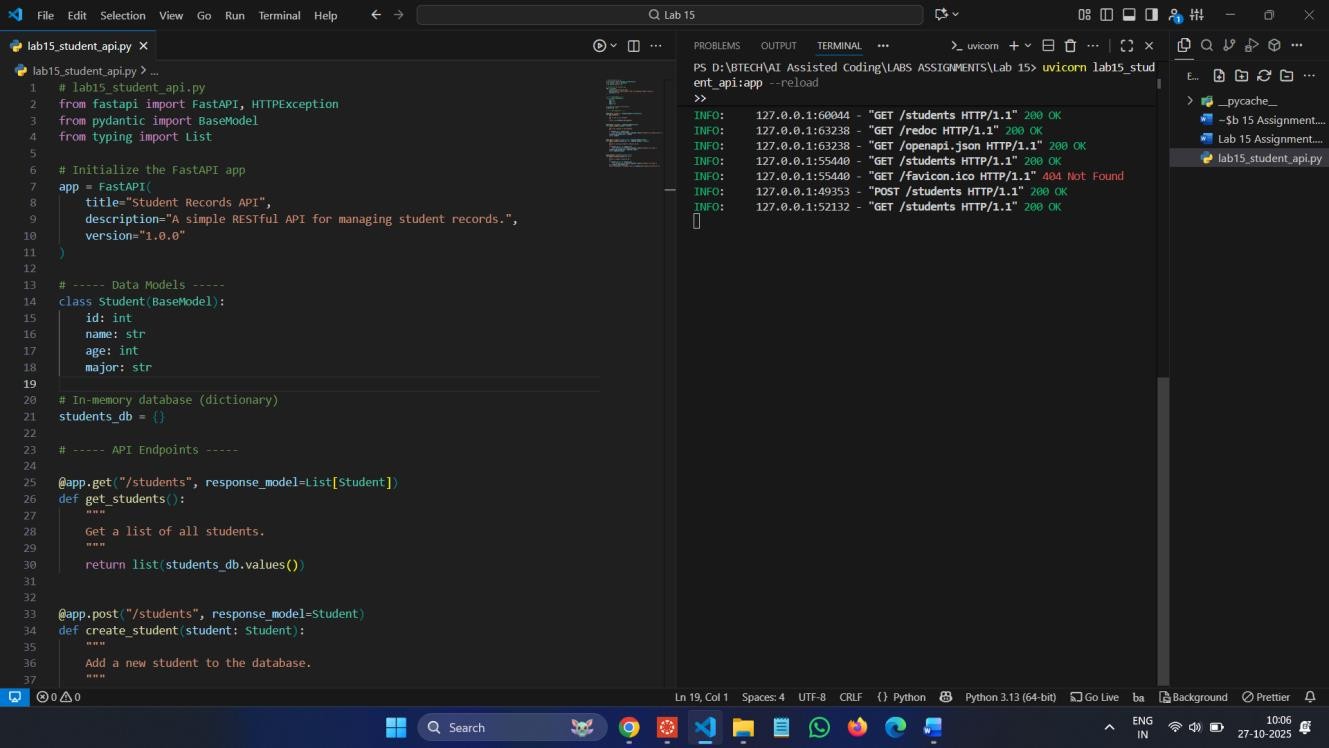
Use AI to build a RESTful API for managing student records.

**Instructions:**

• Endpoints required:

o GET /students → List all students o POST /students → Add a new student o PUT /students/{id} → Update student details o DELETE /students/{id} → Delete a student record

* Use an in-memory data structure (list or dictionary) to store records.
* Ensure API responses are in JSON format. **Code :**



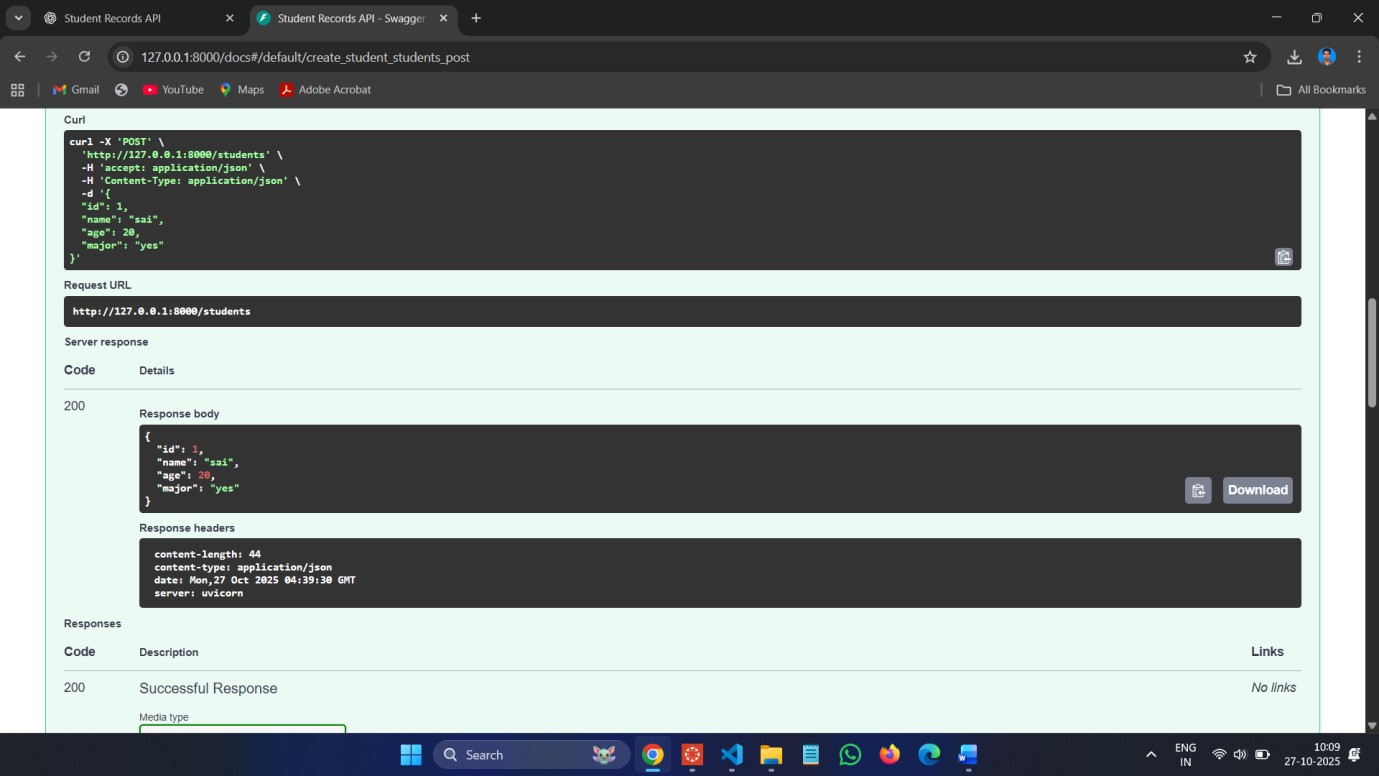
**Swagger UI:** http://127.0.0.1:8000/docs

**Commands :**

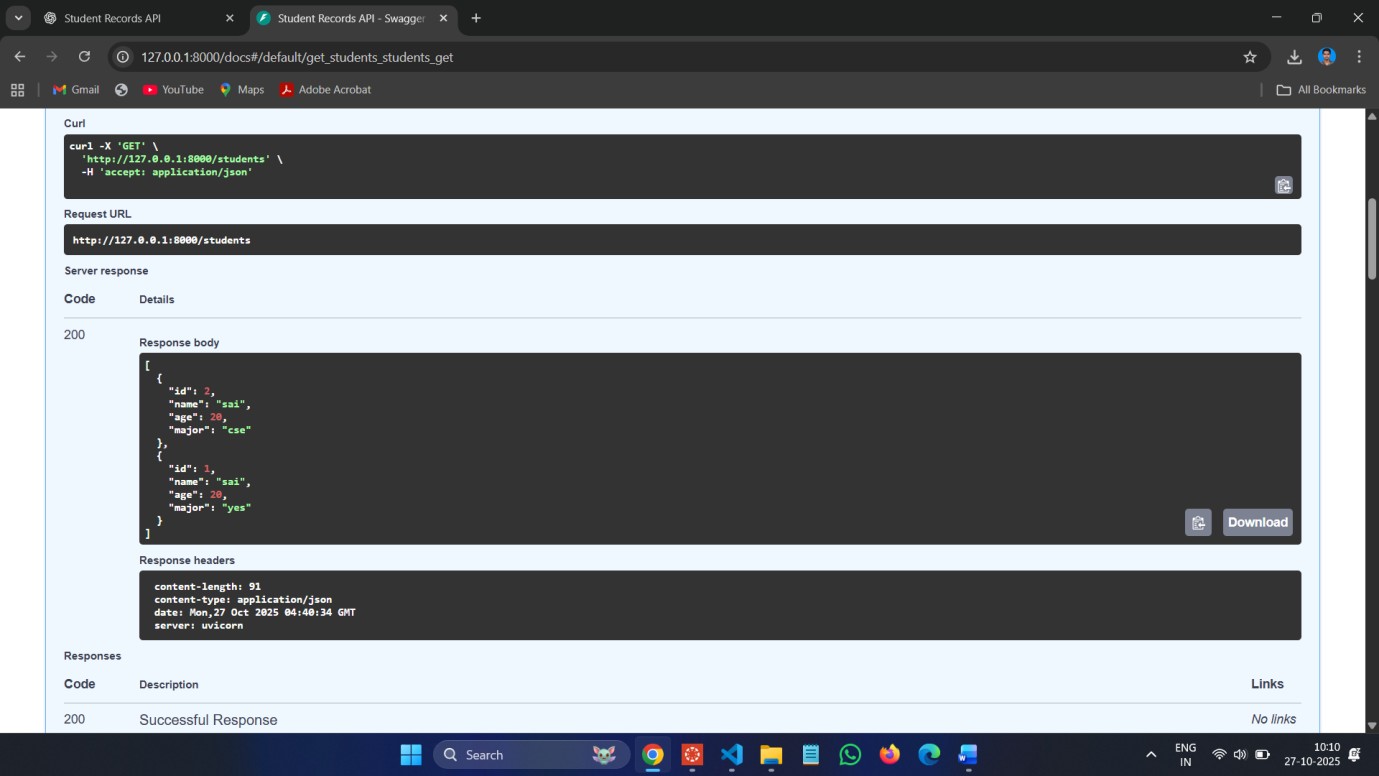
pip install fastapiuvicorn

uvicorn lab15\_student\_api:app –reload **Output:**

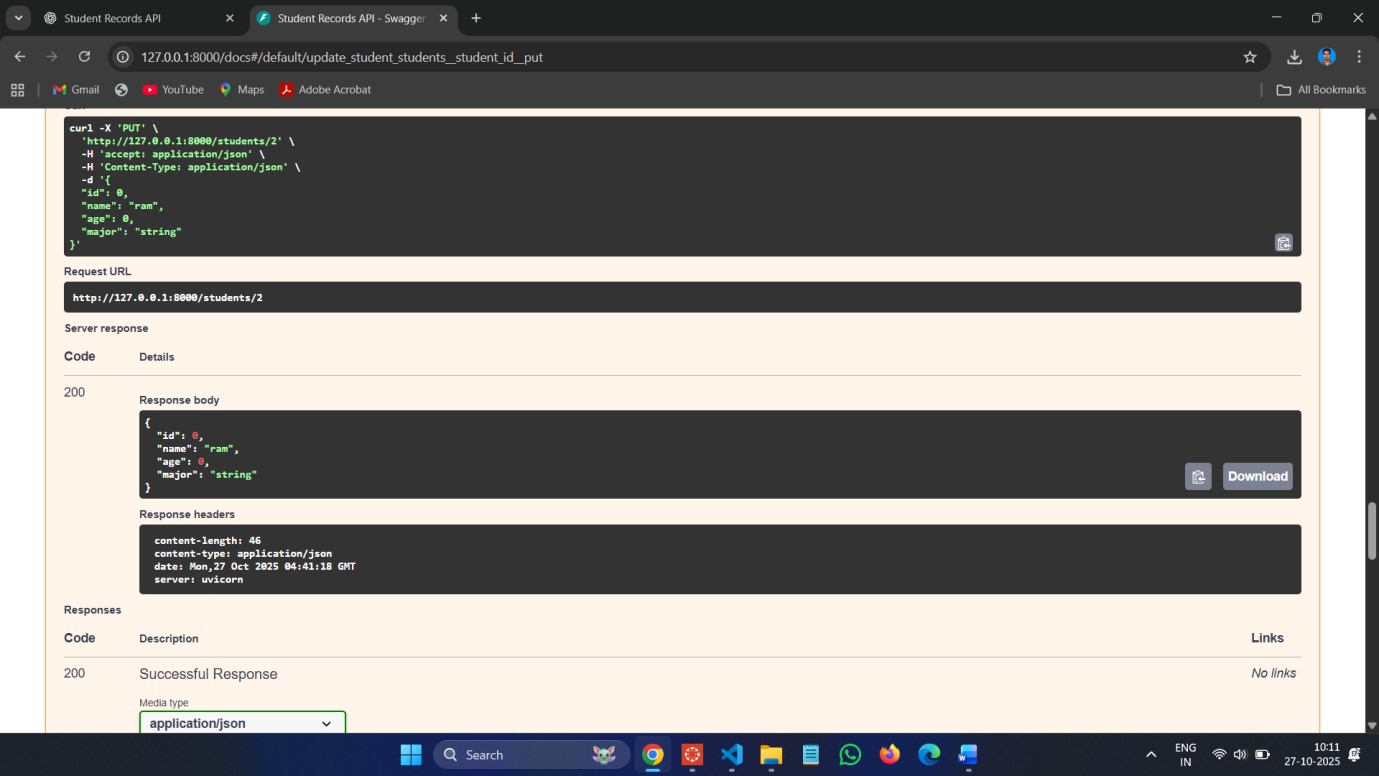
**POST :**



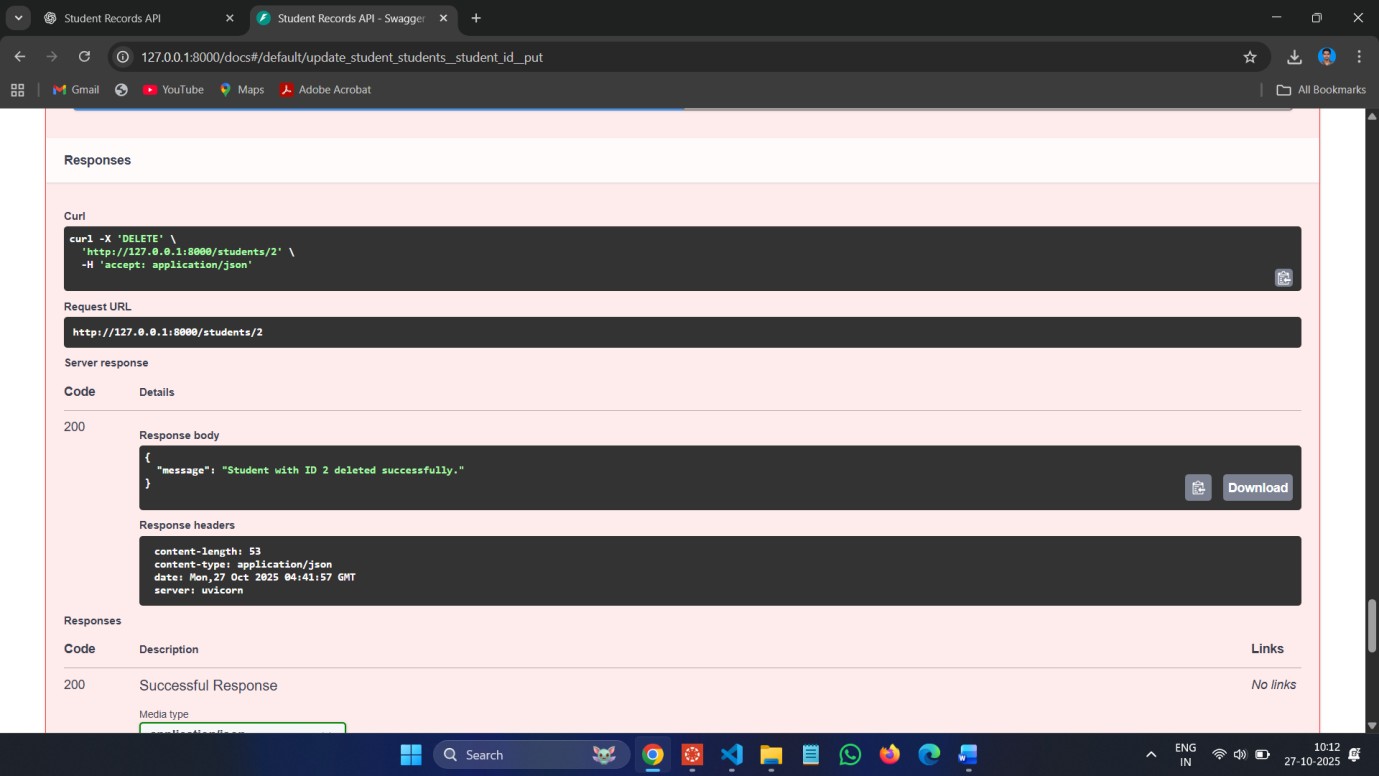
**GET :**



**PUT :**



**DELETE :**



**Task 2 – Library Book Management API Task:**

Develop a RESTful API to handle library books.

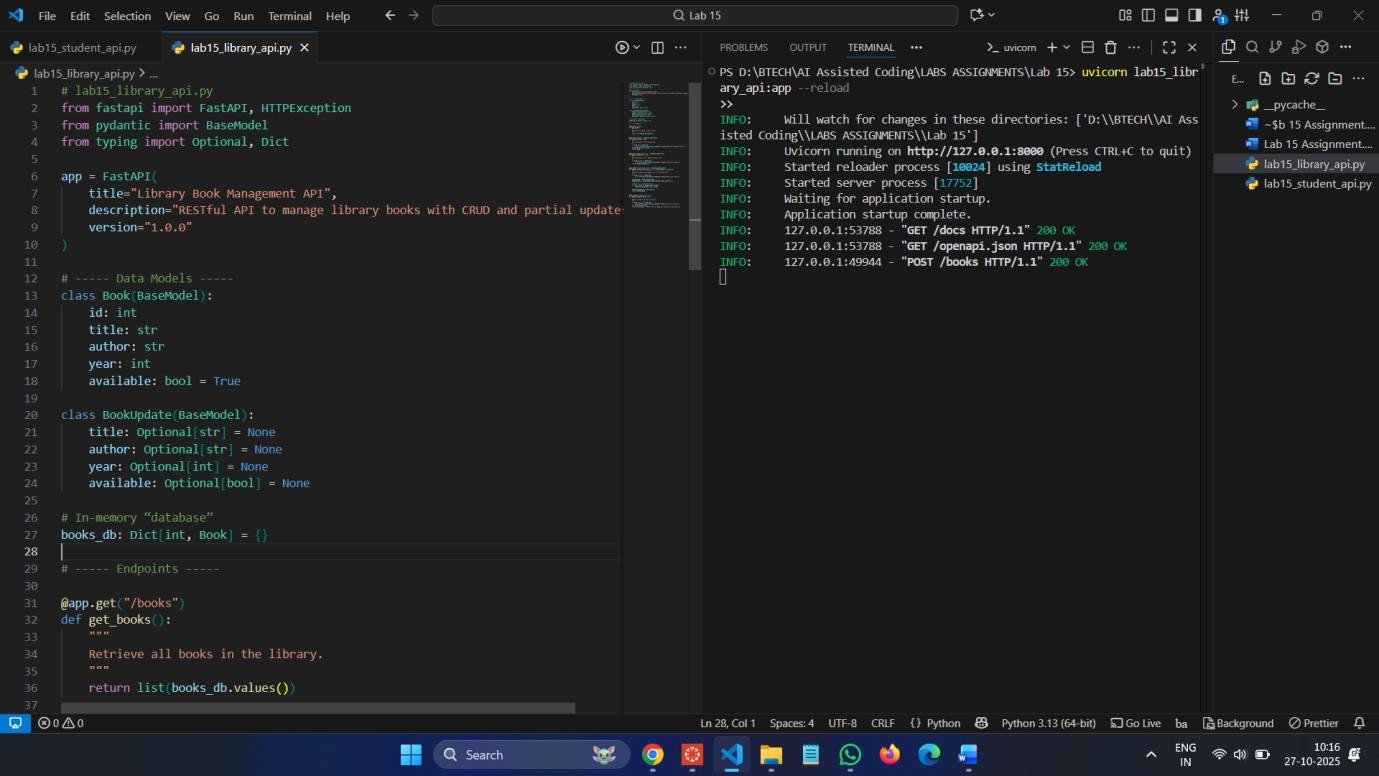
**Instructions:**

• Endpoints required:

* GET /books → Retrieve all books o POST /books → Add a new book
* GET /books/{id} → Get details of a specific book o PATCH /books/{id} → Update partial book details (e.g., availability)
* DELETE /books/{id} → Remove a book

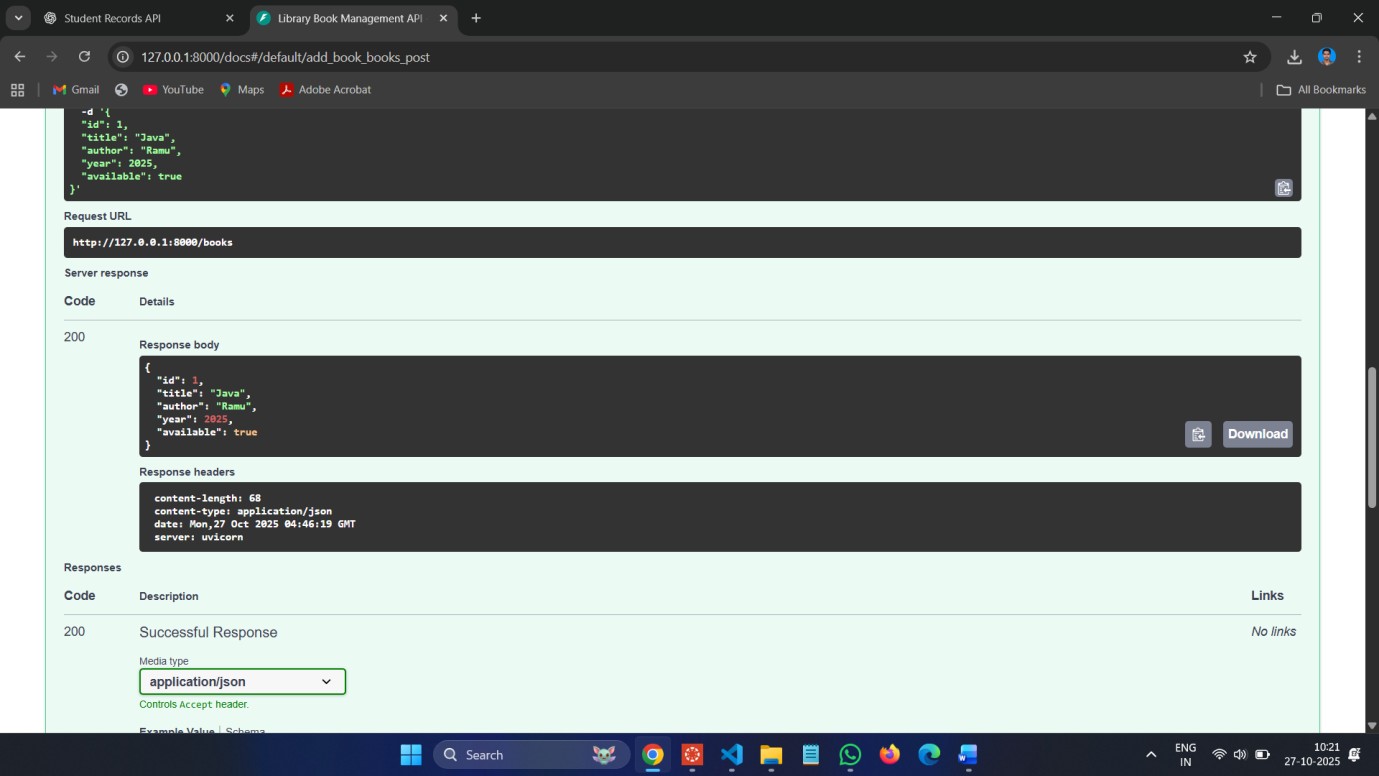
• Implement error handling for invalid requests.

**Code :**

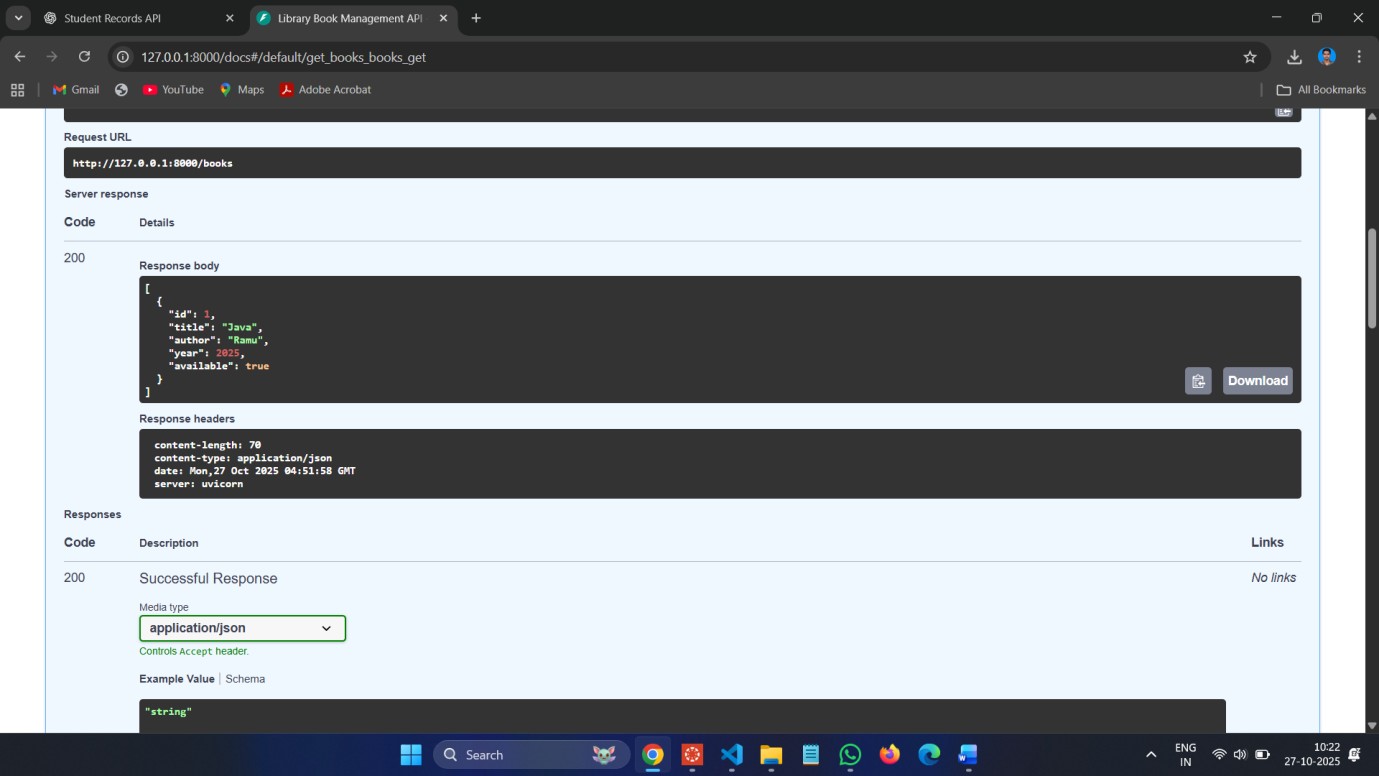


**Commands :**

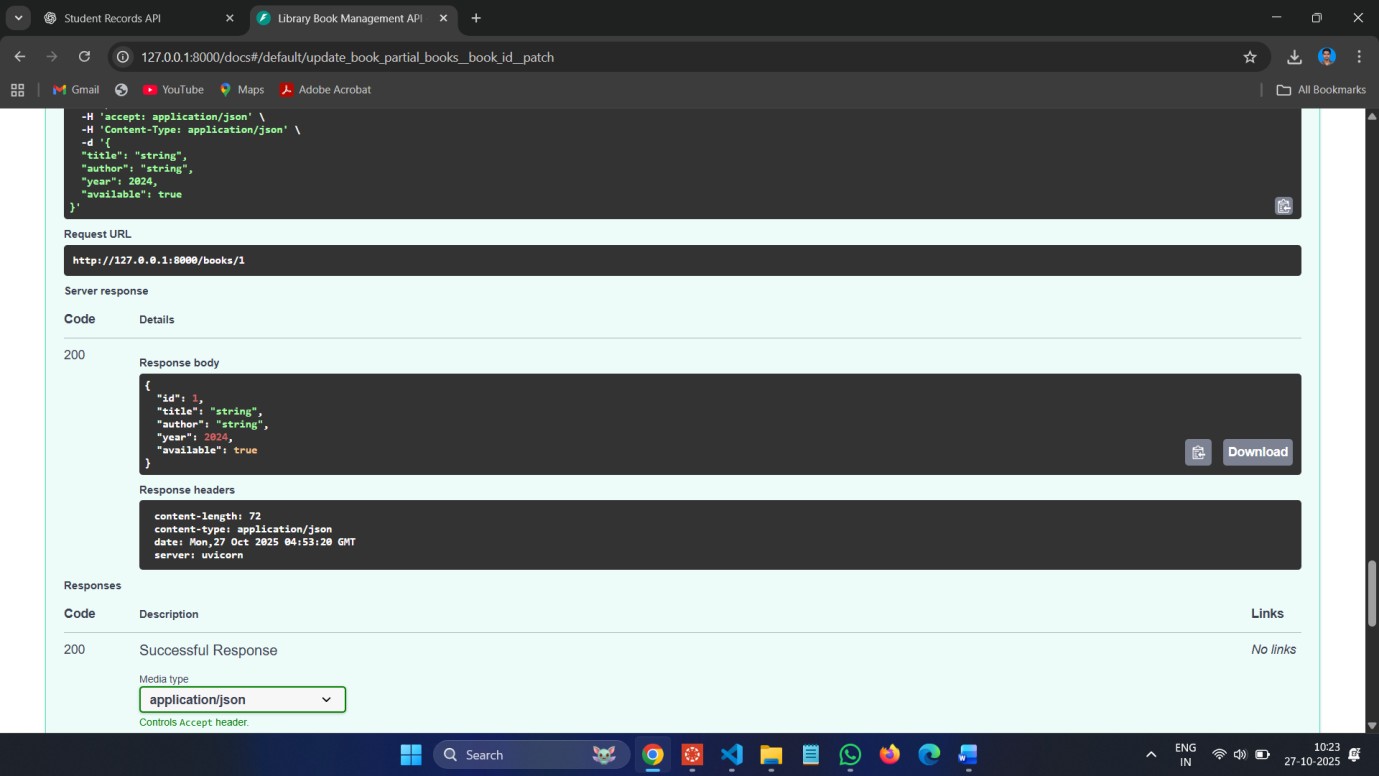
uvicorn lab15\_library\_api:app –reload **Swagger UI:**[**http://127.0.0.1:8000/docs**](http://127.0.0.1:8000/docs) **POST :**



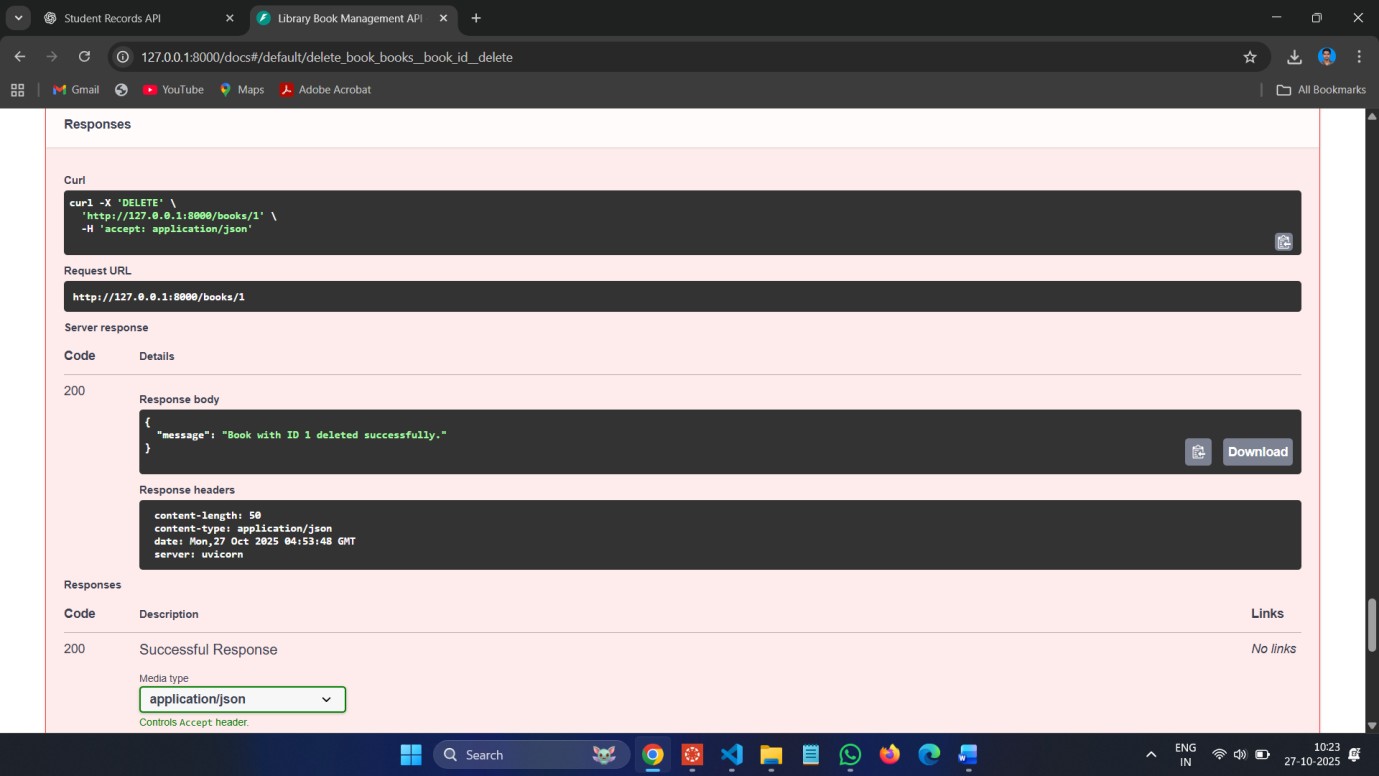
**GET :**



**PUT :**



**DELETE :**



**Task 3 – Employee Payroll API Task:**

Create an API for managing employees and their salaries.

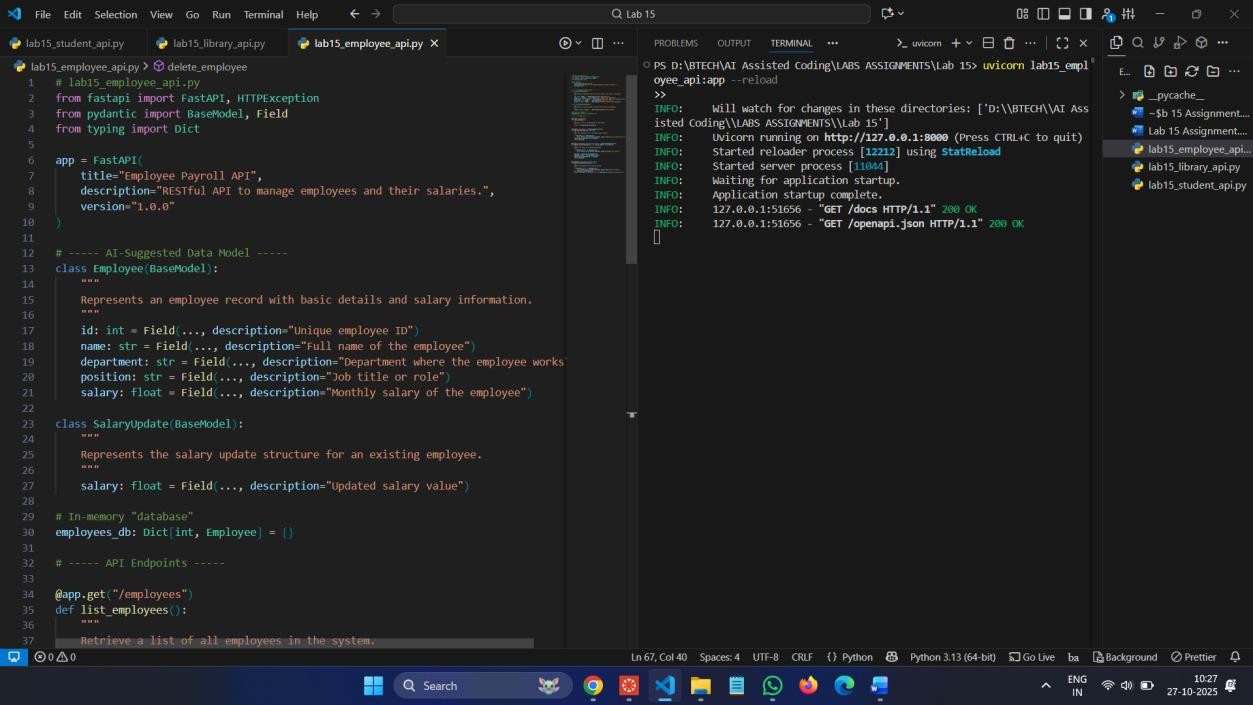
**Instructions:**

• Endpoints required:

* GET /employees → List all employees
* POST /employees → Add a new employee with salary details
* PUT /employees/{id}/salary → Update salary of an employee
* DELETE /employees/{id} → Remove employee from system

**• Use AI to:**

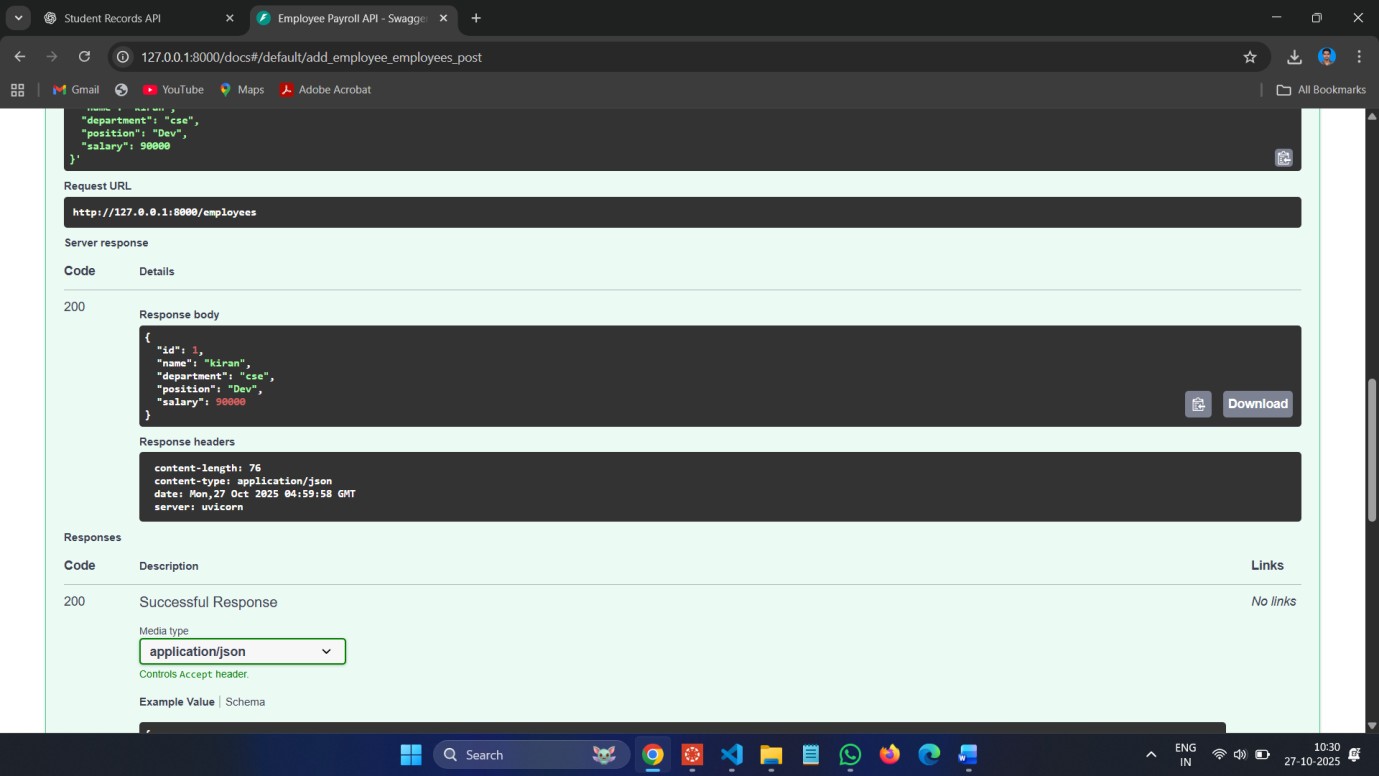
* Suggest data model structure.
* Add comments/docstrings for all endpoints. **Code :**



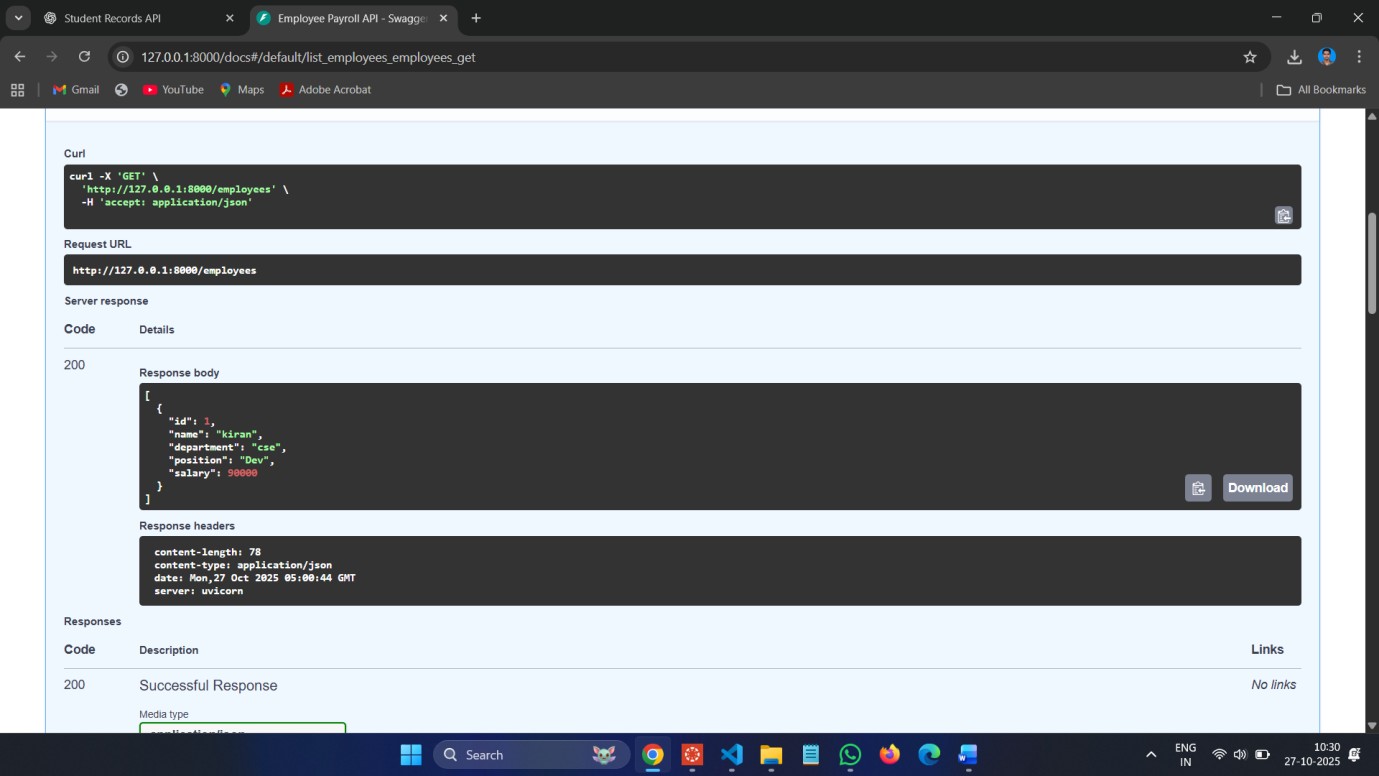
**Commands :**

**uvicorn lab15\_employee\_api:app –reload**

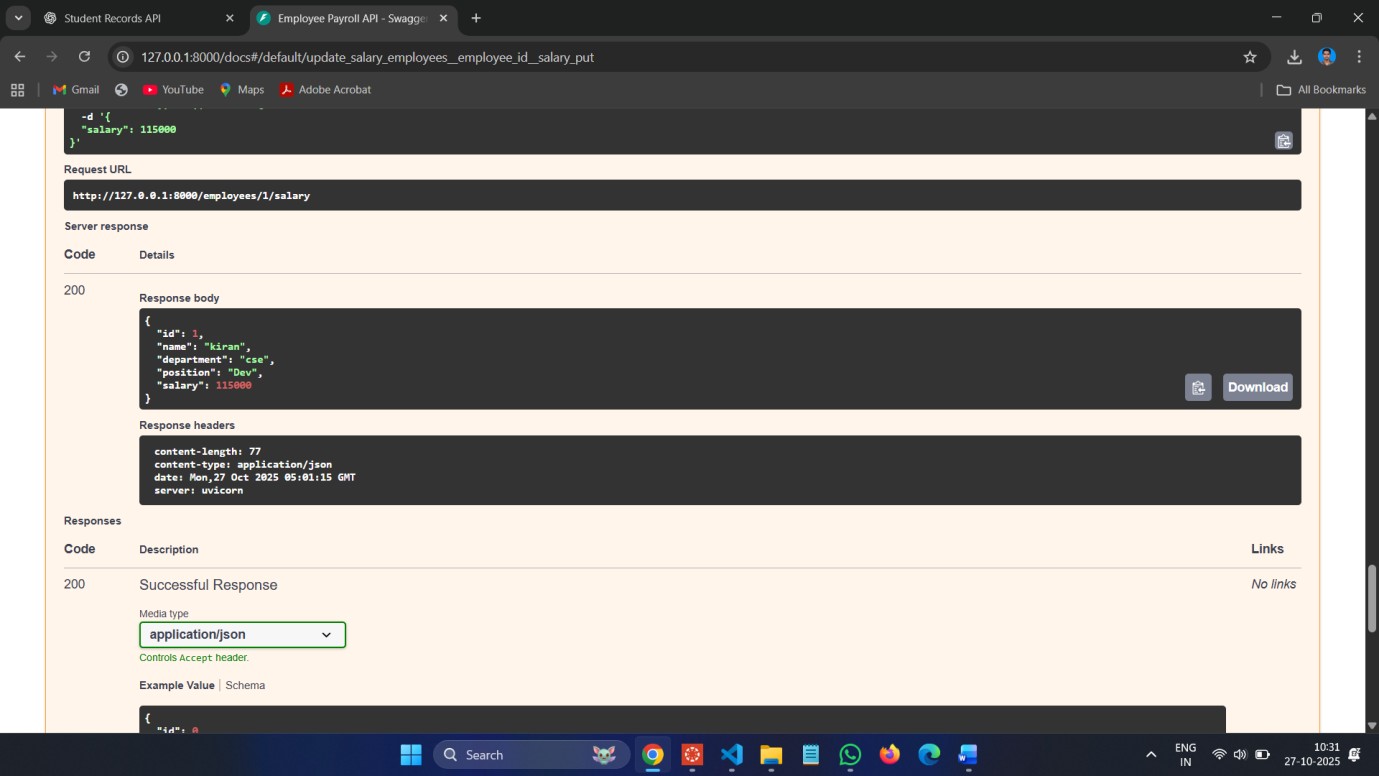
**Swagger UI:** [**http://127.0.0.1:8000/docs**](http://127.0.0.1:8000/docs) **POST :**



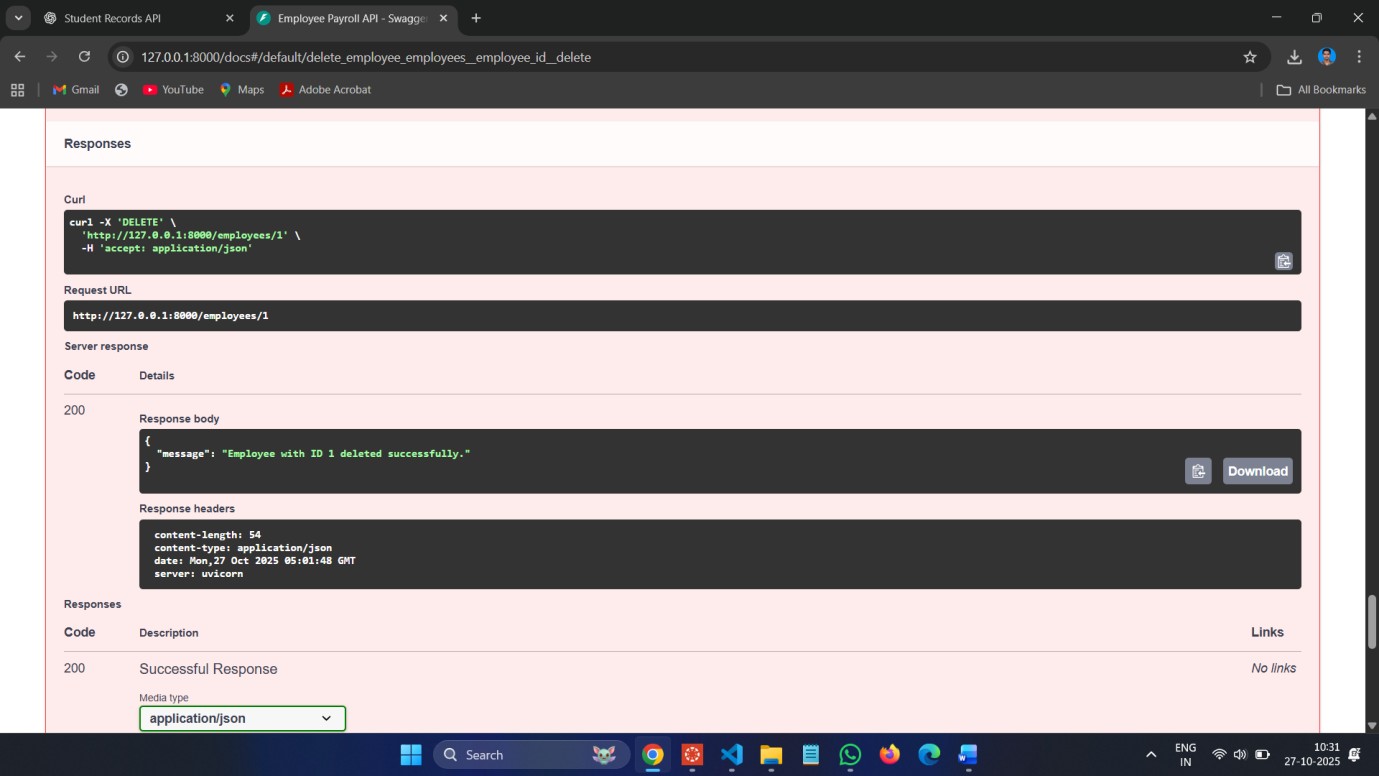
**GET :**



**PUT :**



**DELETE :**



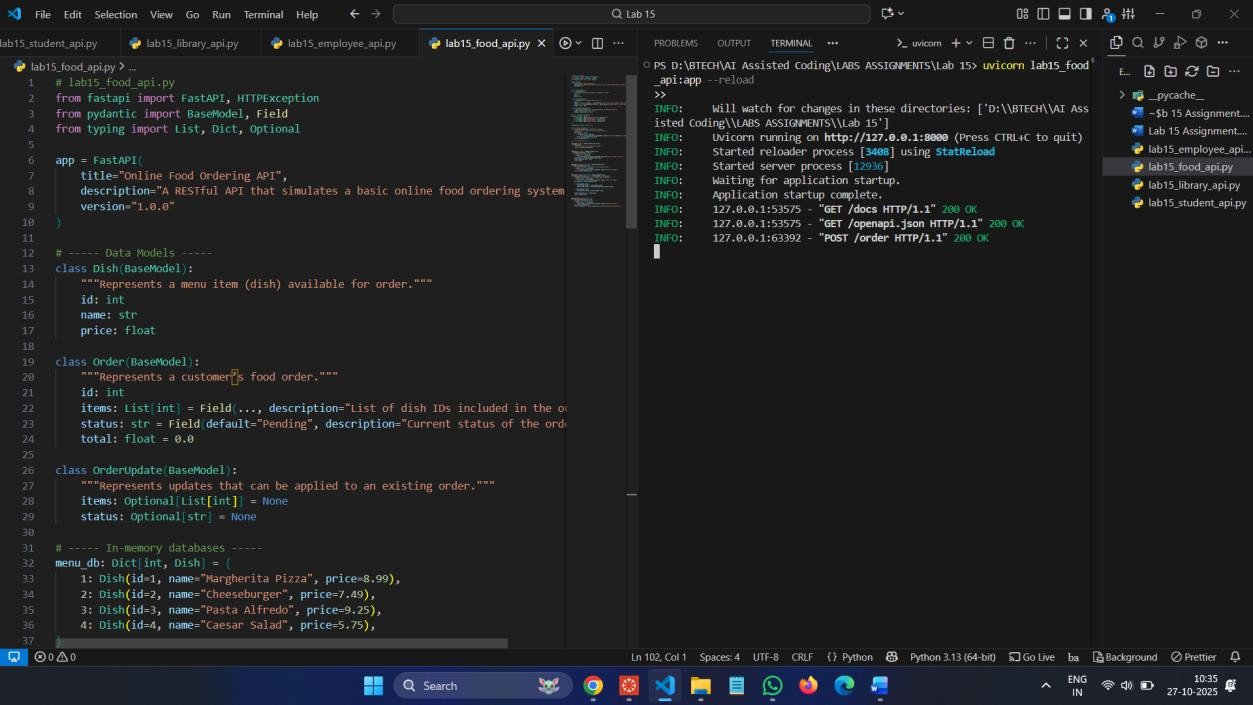
**Task 4 – Real-Time Application: Online Food Ordering API Scenario:**

Design a simple API for an online food ordering system.

**Requirements:**

**• Endpoints required:**

* GET /menu → List available dishes o POST /order → Place a new order o GET /order/{id} → Track order status o PUT /order/{id} → Update an existing order (e.g., change items) o DELETE /order/{id} → Cancel an order **• AI should generate:**
* REST API code
* Suggested improvements (like authentication, pagination) **Code :**

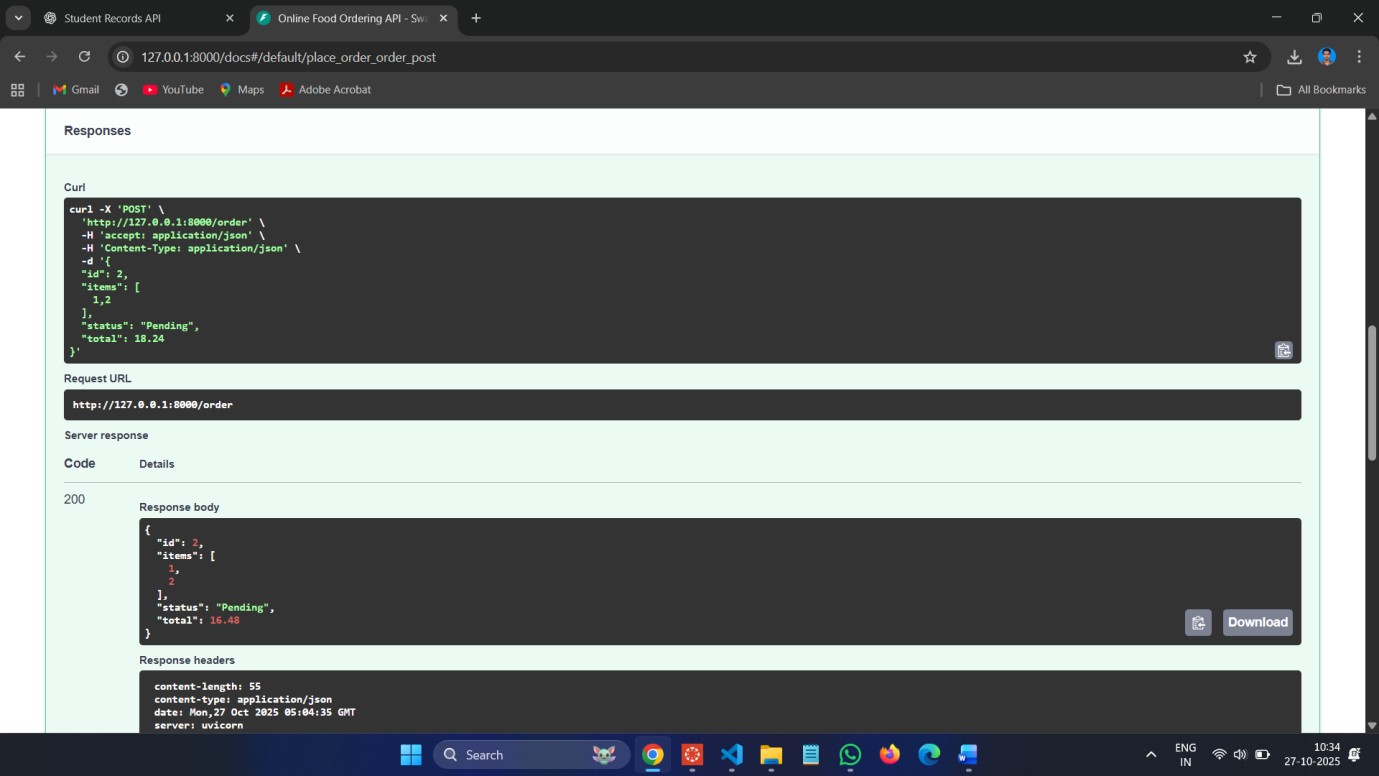


**Commands :**

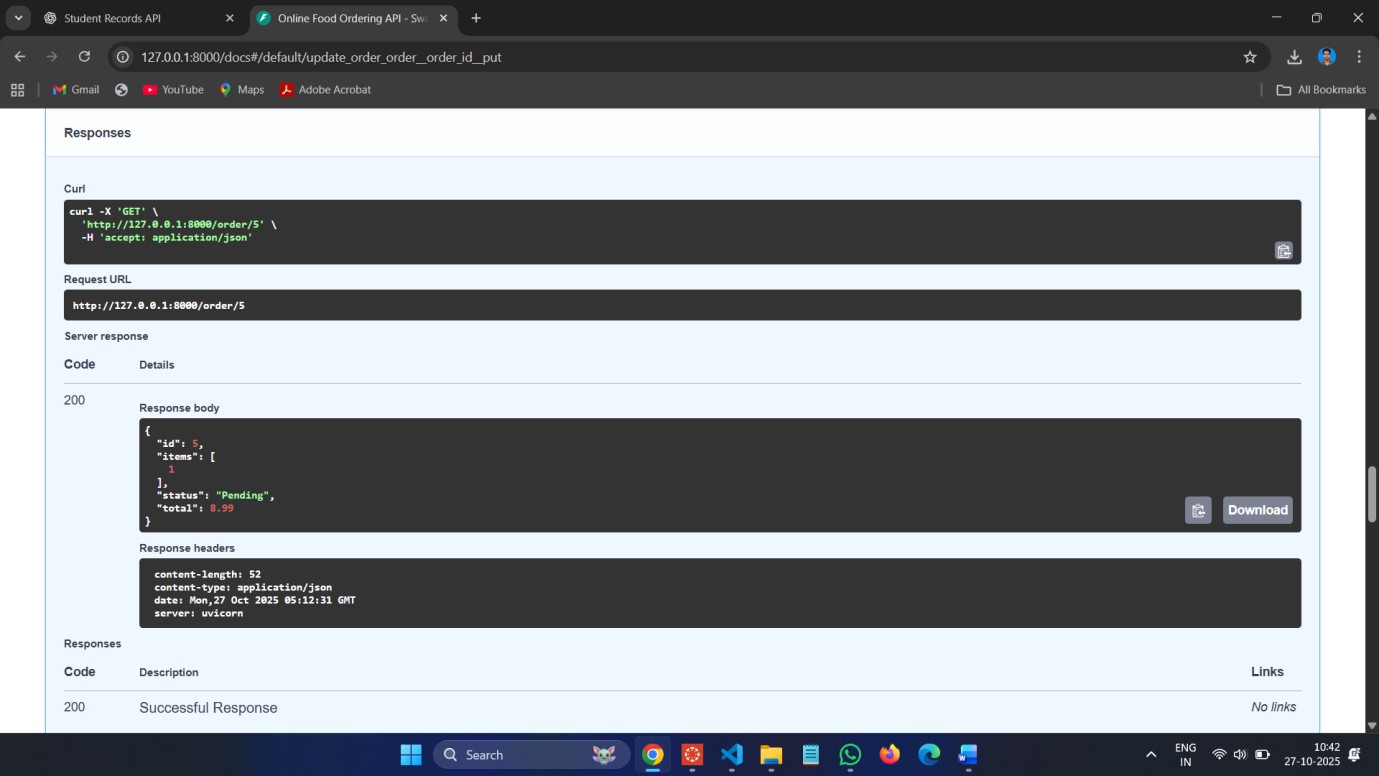
**uvicorn lab15\_food\_api:app --reload**

**Swagger UI:** [**http://127.0.0.1:8000/docs**](http://127.0.0.1:8000/docs)

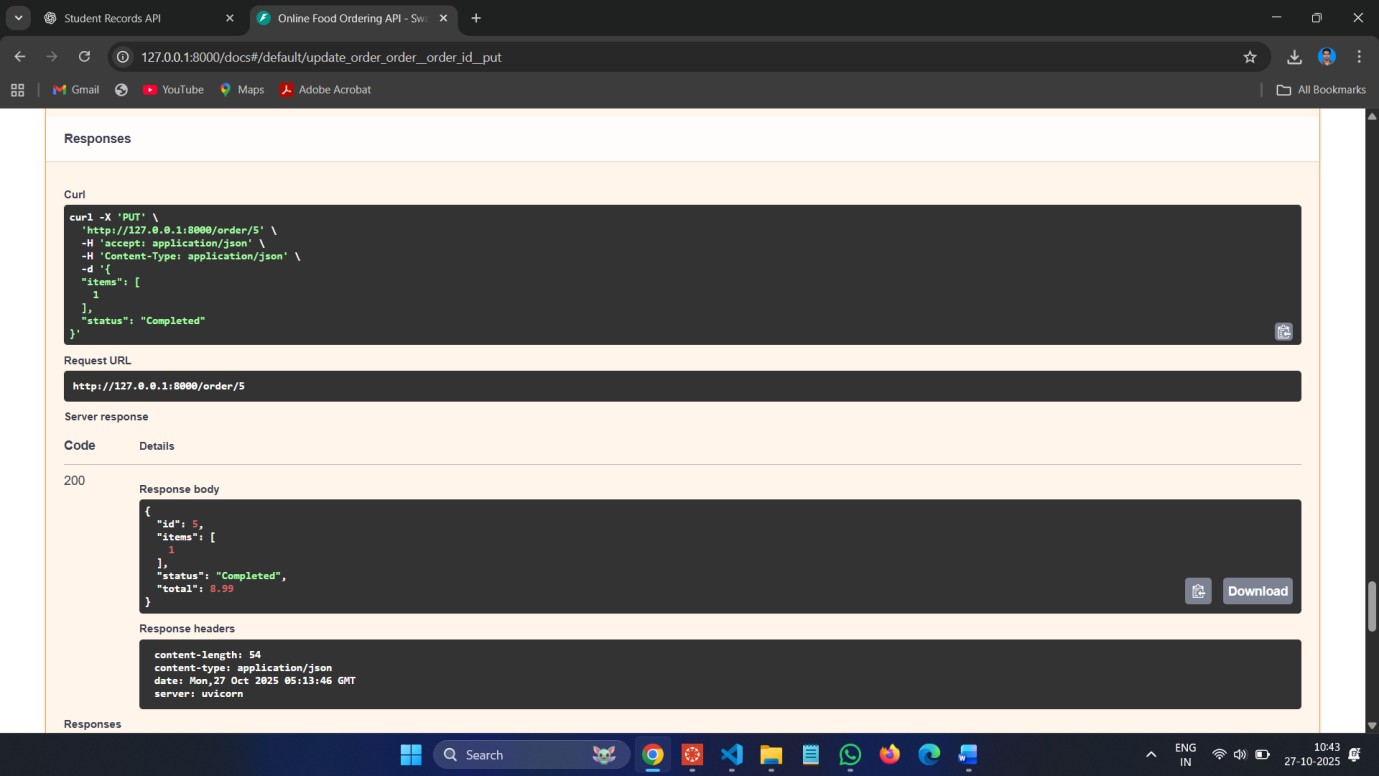
**POST :**



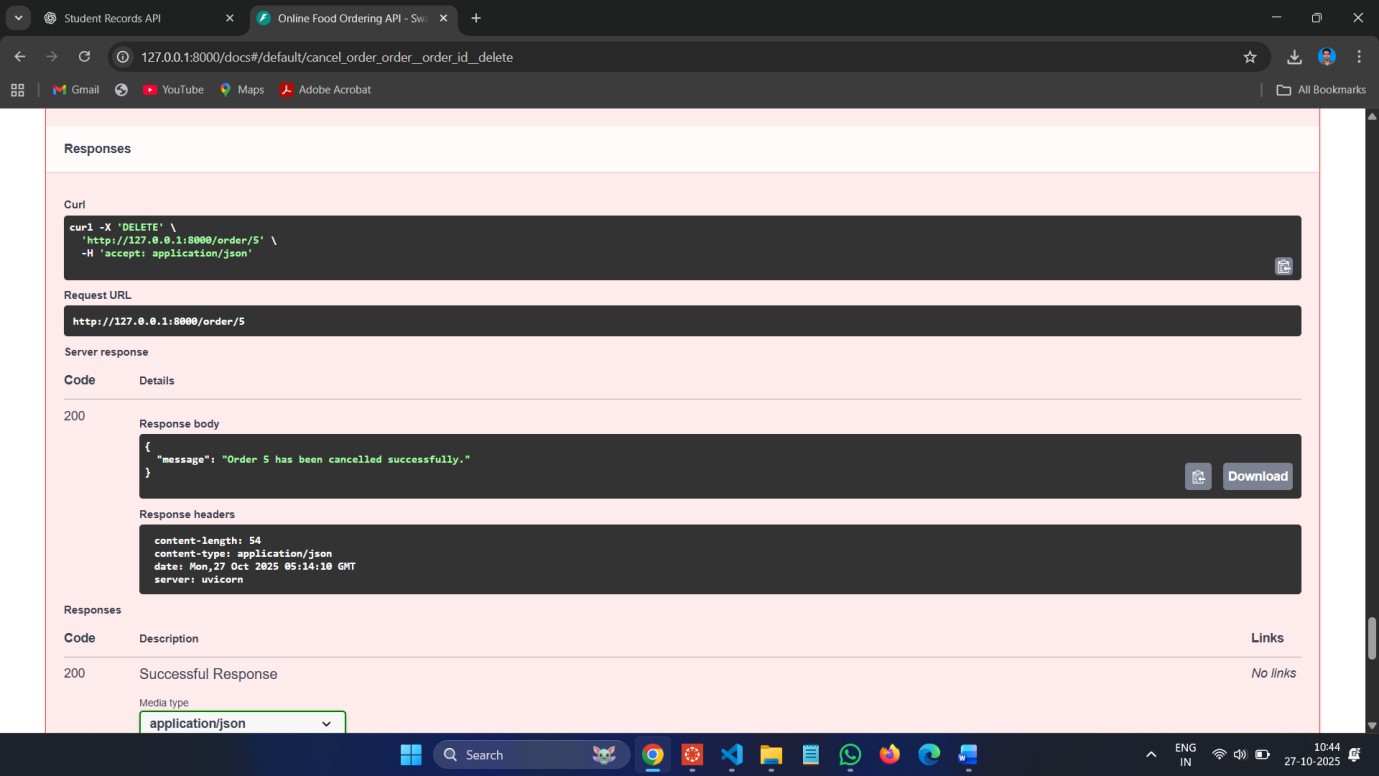
**GET :**



**PUT :**



**DELETE :**



**Observations:**

* 1. **RESTful Design Principles:**

Each task followed REST standards — using appropriate HTTP methods (GET, POST, PUT, PATCH, DELETE) and clear URL structures for resources like /students, /books, /employees, and /order.

* 1. **AI-Assisted Code Generation:**

AI tools were effectively used to generate data models, endpoint structures, and documentation automatically, saving development time and ensuring consistency.

* 1. **CRUD Operations:**

All four APIs successfully implemented CRUD functionality:

* Create: POST requests o Read: GET requests o Update: PUT/PATCH requests o Delete: DELETE requests

1. **Error Handling:**

Each API handled invalid requests gracefully using HTTPException with proper status codes and messages.

1. **Auto Documentation:**

FastAPI automatically provided API documentation through /docs (Swagger UI) and /redoc (ReDoc), fulfilling the lab’s documentation objective.

1. **Partial Updates & Real-Time Design:** 
   * Task 2 introduced partial updates (PATCH) for library books.
   * Task 4 simulated a real-world food ordering system with realistic order tracking and AI improvement suggestions.
2. **AI-Driven Improvements:**

Suggested features such as authentication, pagination, realtime updates (via WebSockets), and database integration demonstrate how AI can enhance backend architecture planning.