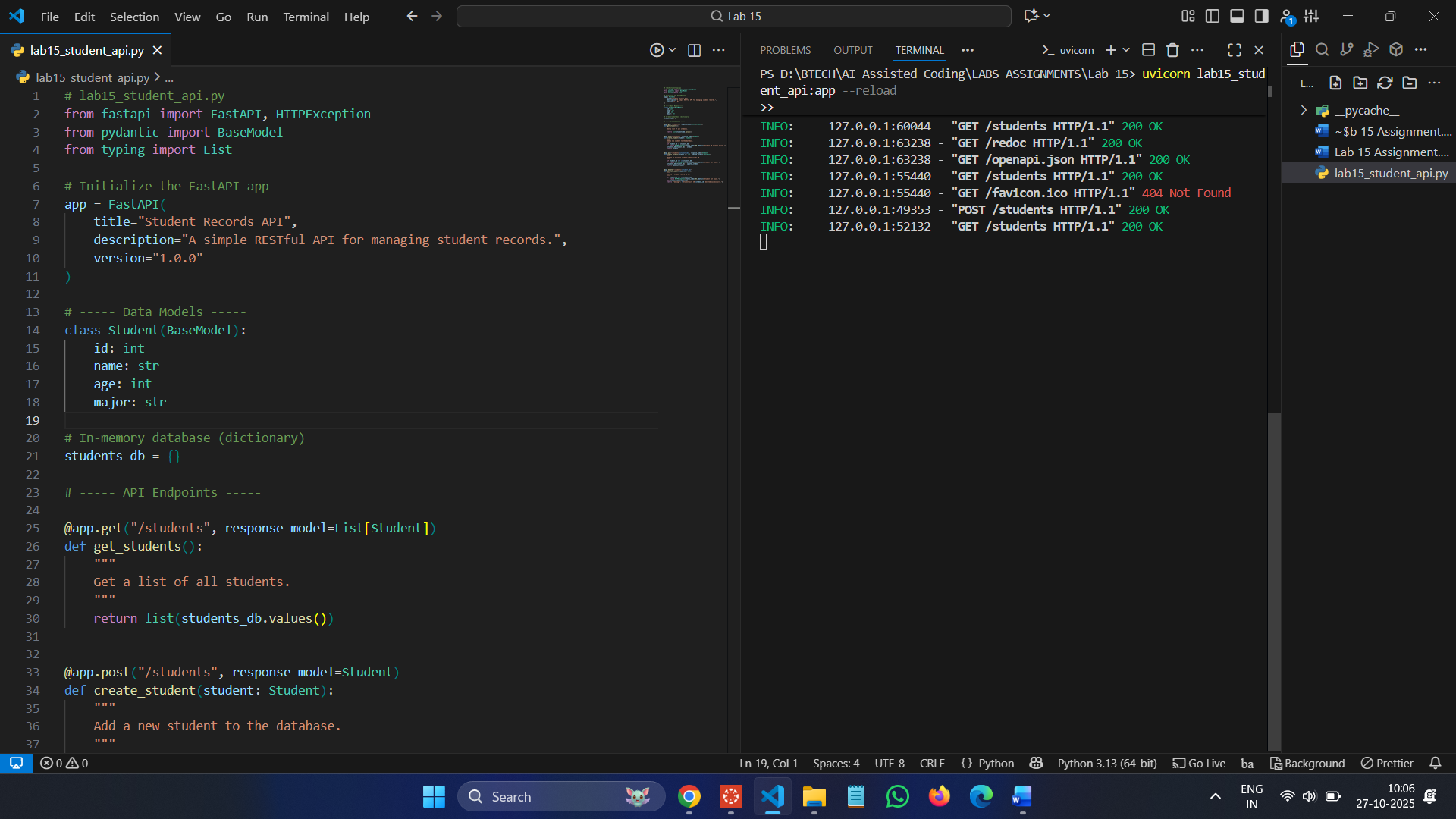
**Lab 15 – Backend API Development: Creating RESTful Services with AI**

**Htno : 2503A52L14**

**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

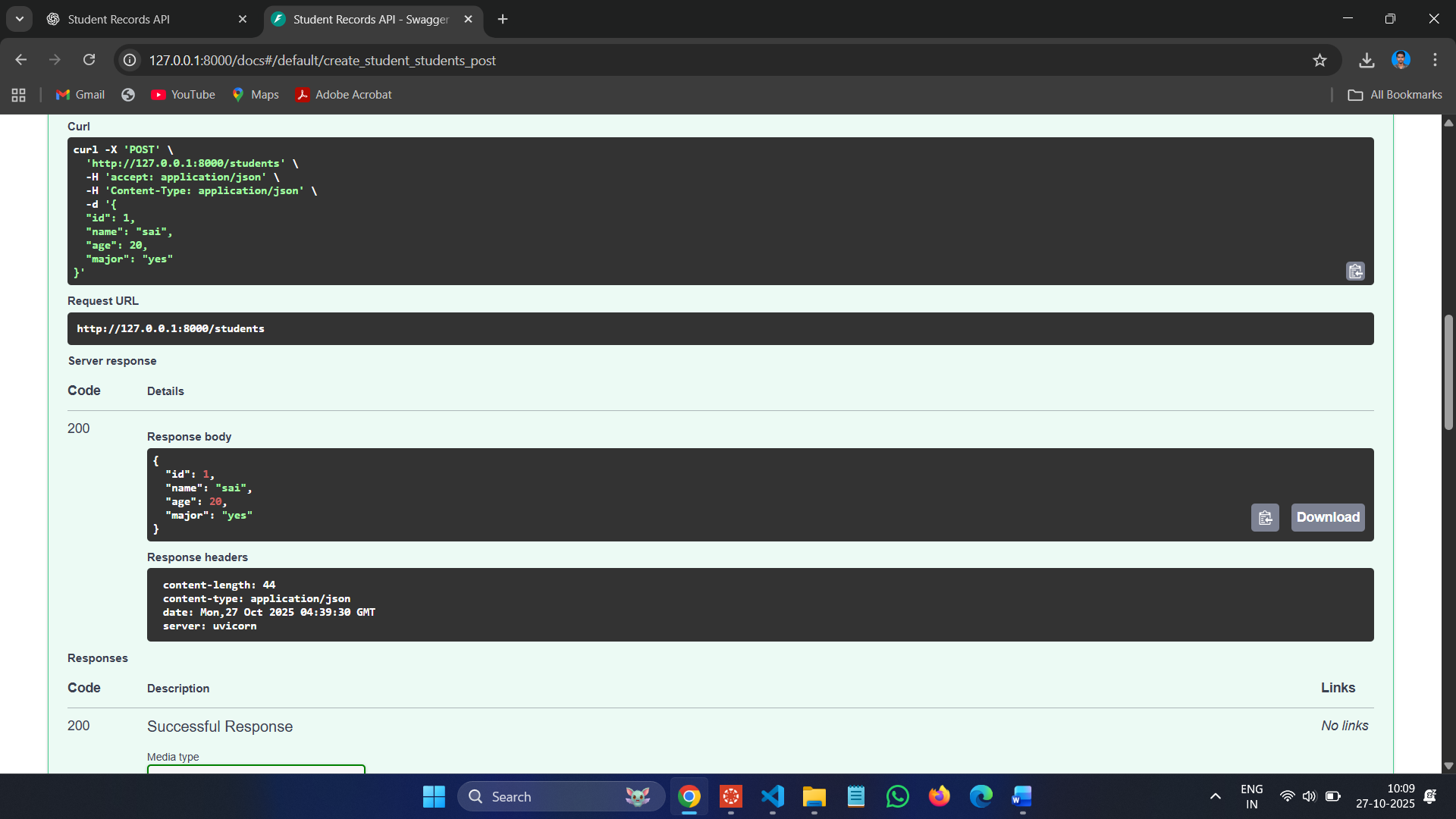
**Commands :**

pip install fastapi uvicorn

uvicorn lab15\_student\_api:app –reload

**Output :**

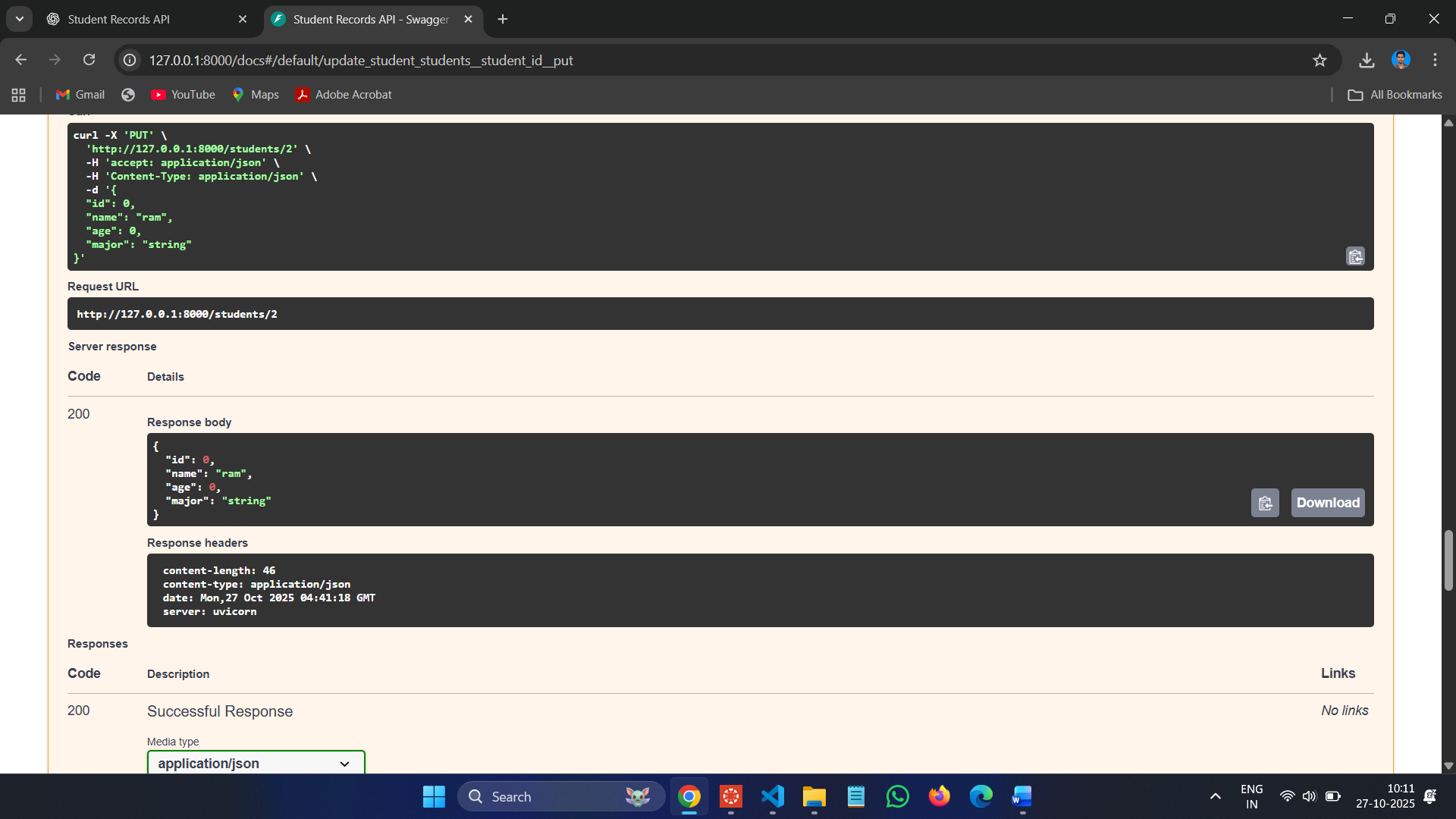
**POST :**



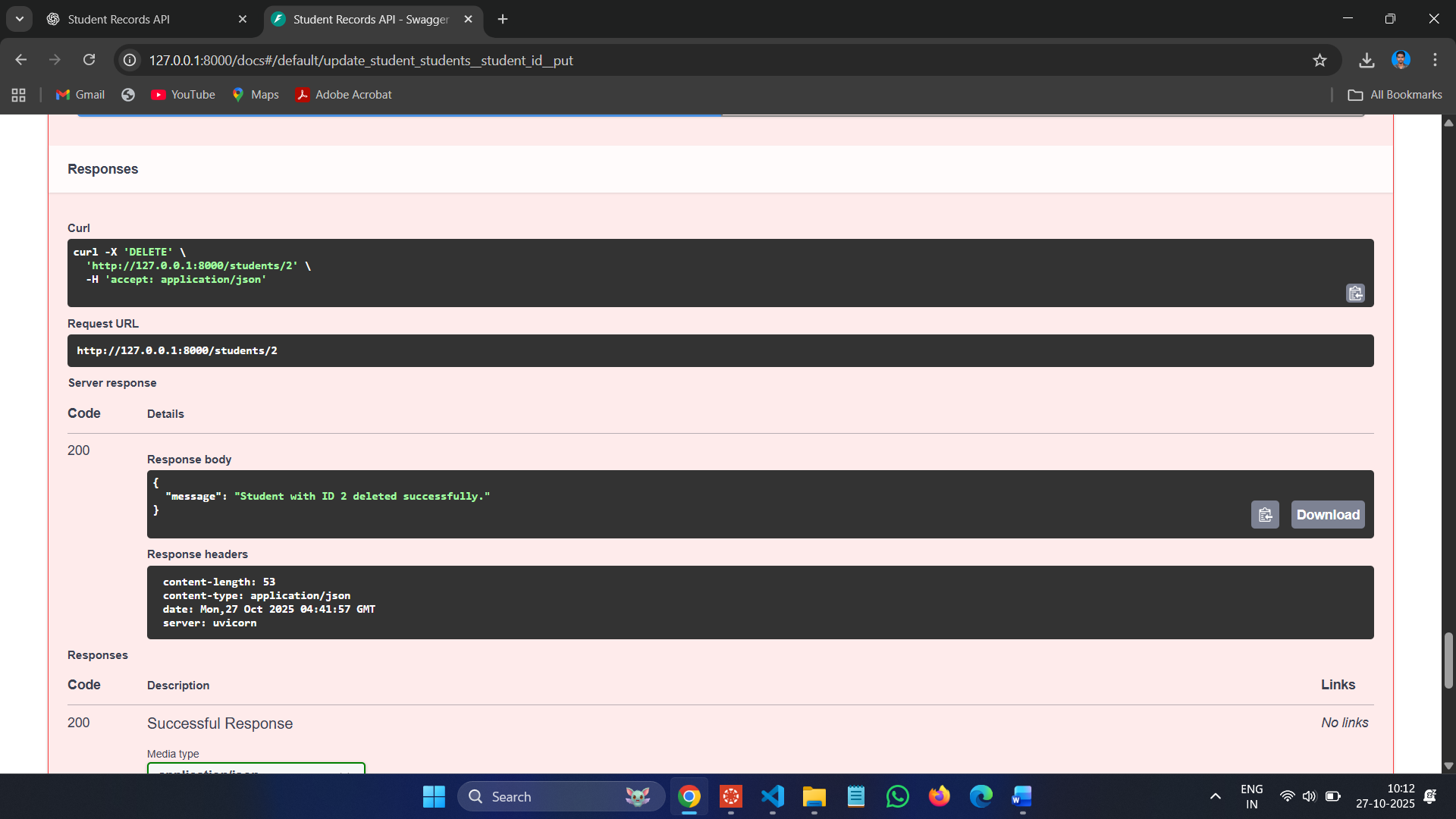
**GET :**



**PUT :**

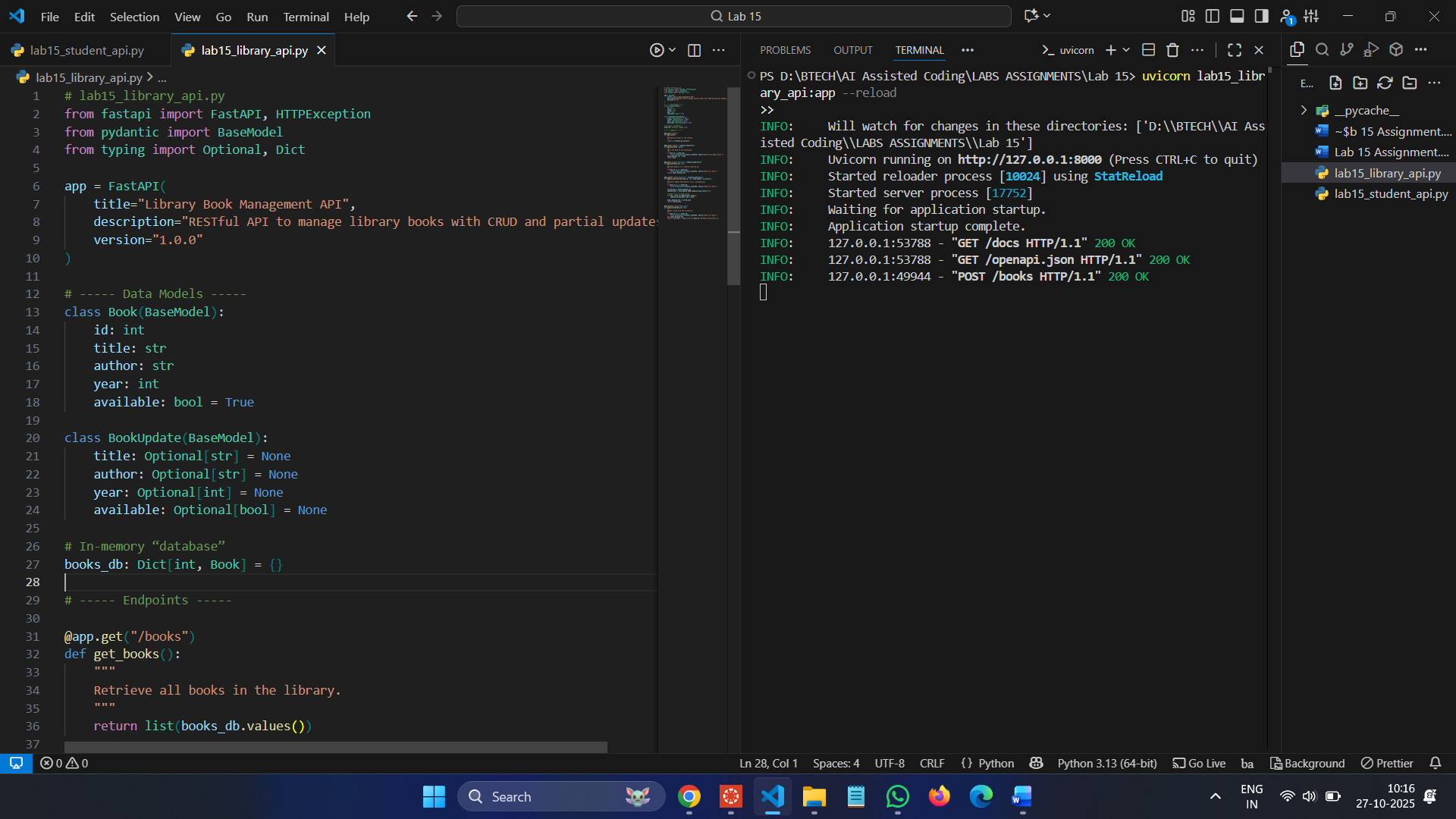


**DELETE :**



**Task 2 – Library Book Management API  
Task:**  
Develop a RESTful API to handle library books.  
**Instructions:**  
• Endpoints required:  
o GET /books → Retrieve all books  
o POST /books → Add a new book  
o GET /books/{id} → Get details of a specific book  
o PATCH /books/{id} → Update partial book details (e.g.,  
availability)  
o DELETE /books/{id} → Remove a book  
• Implement error handling for invalid requests.

**Code :**

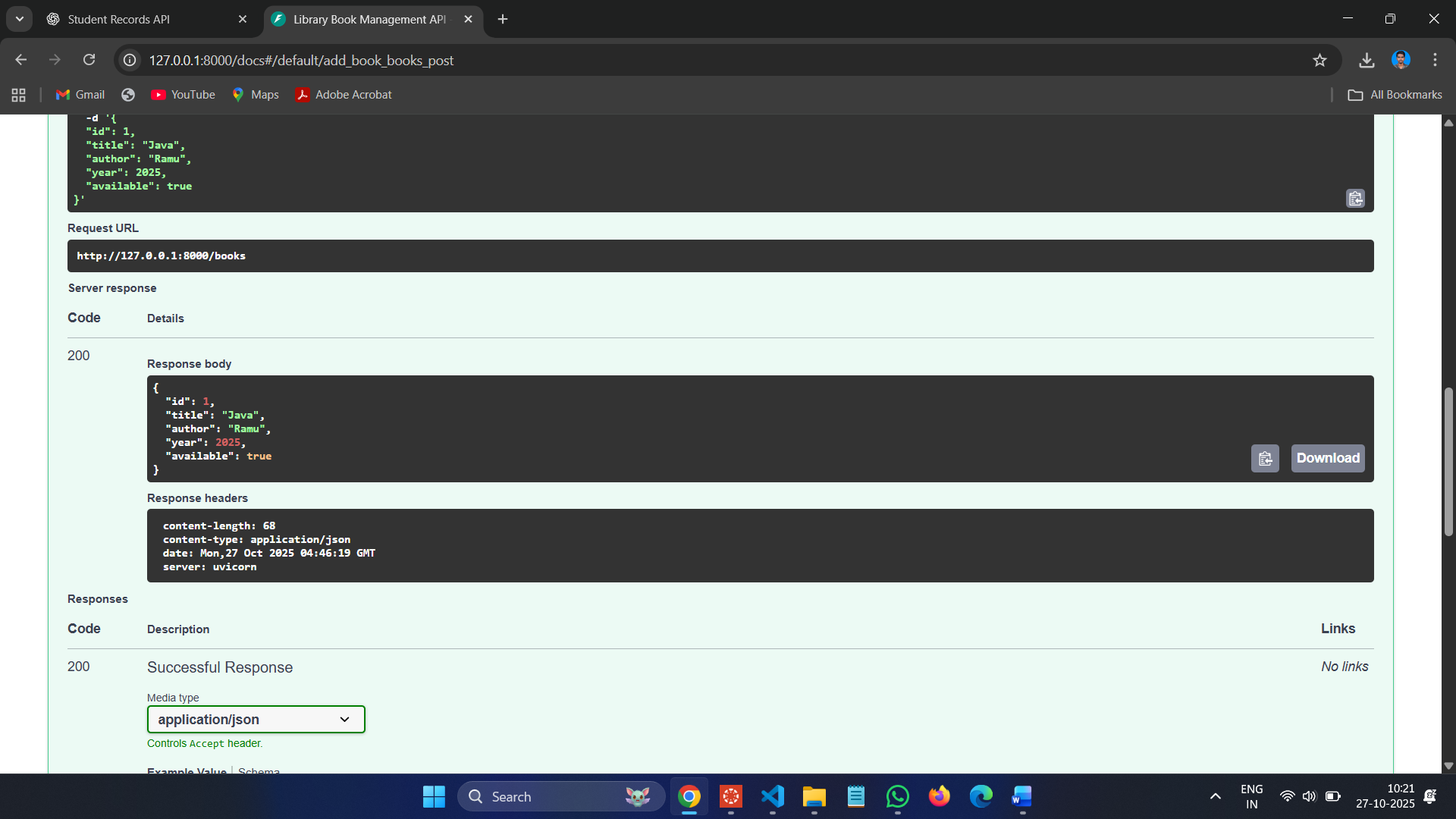


**Commands :**

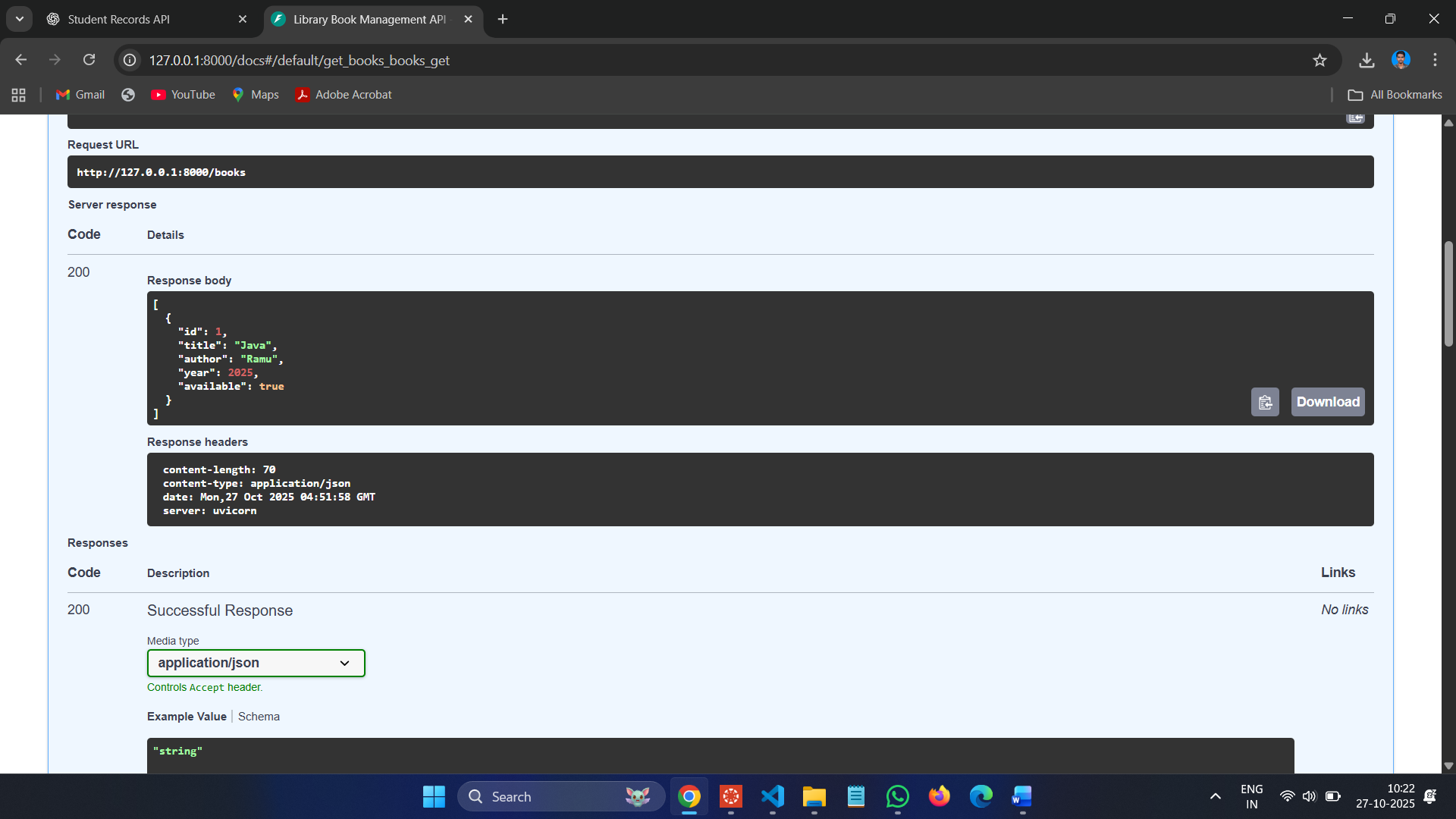
uvicorn lab15\_library\_api:app –reload

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

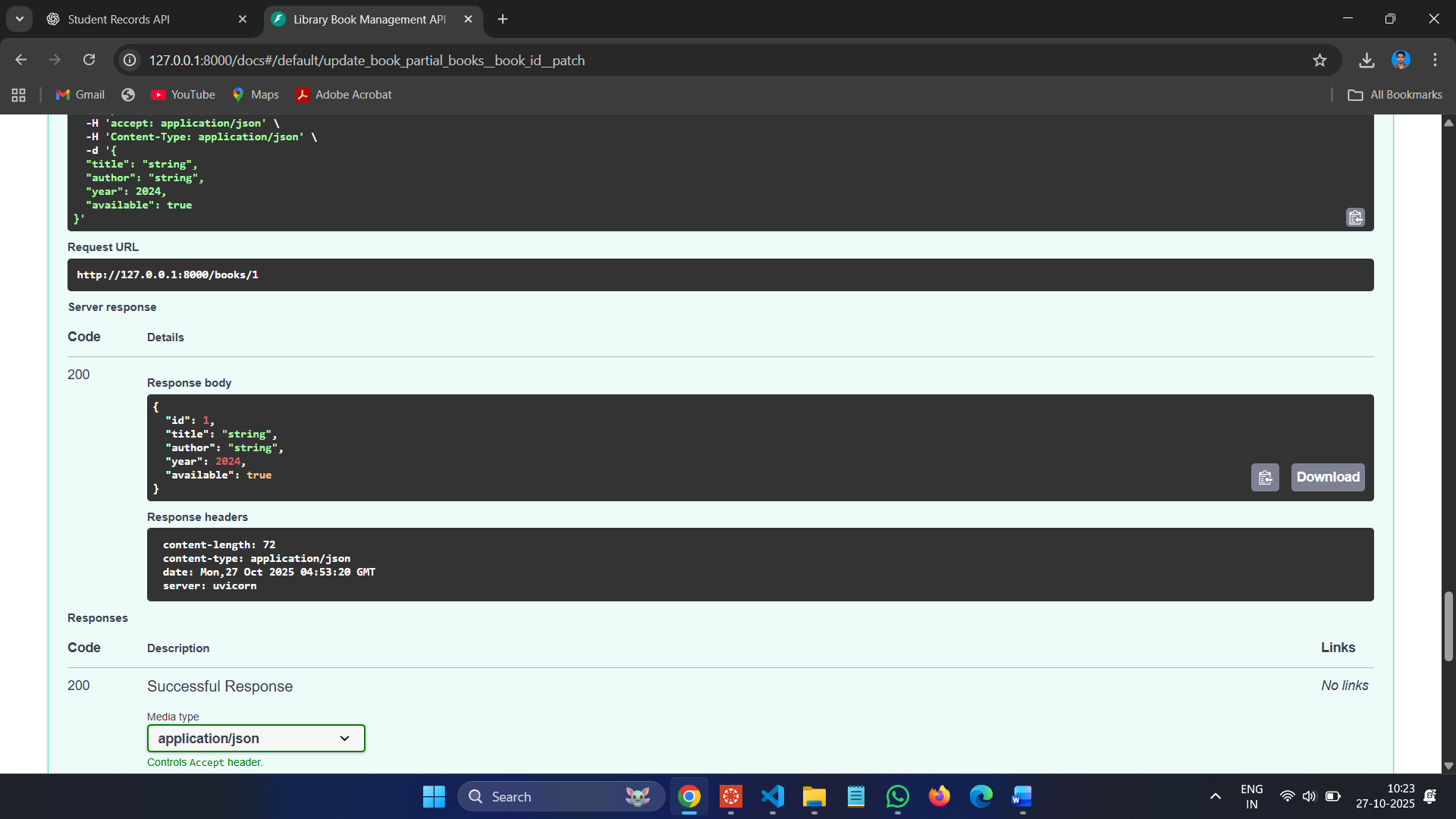
**POST :**



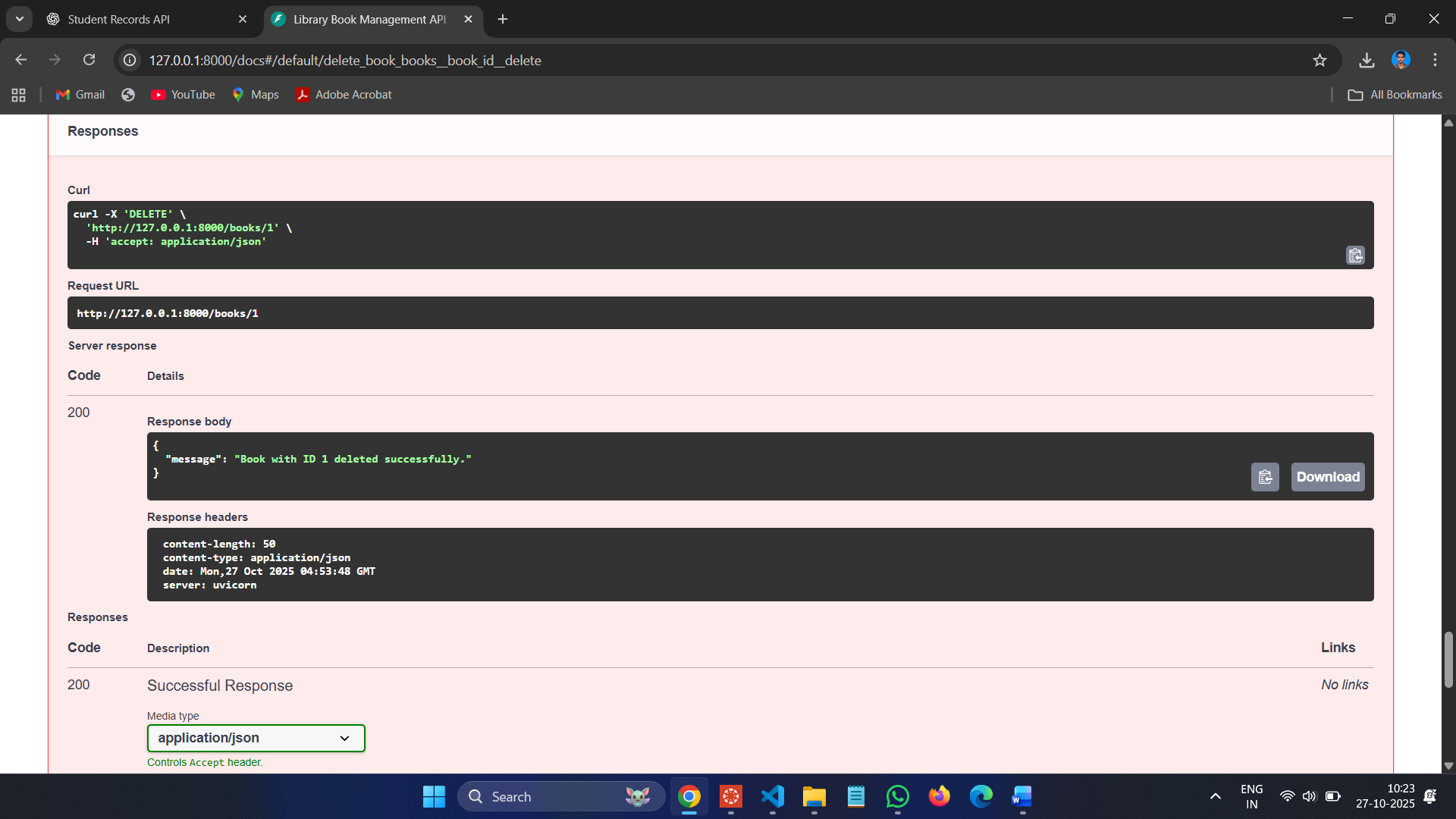
**GET :**



**PUT :**

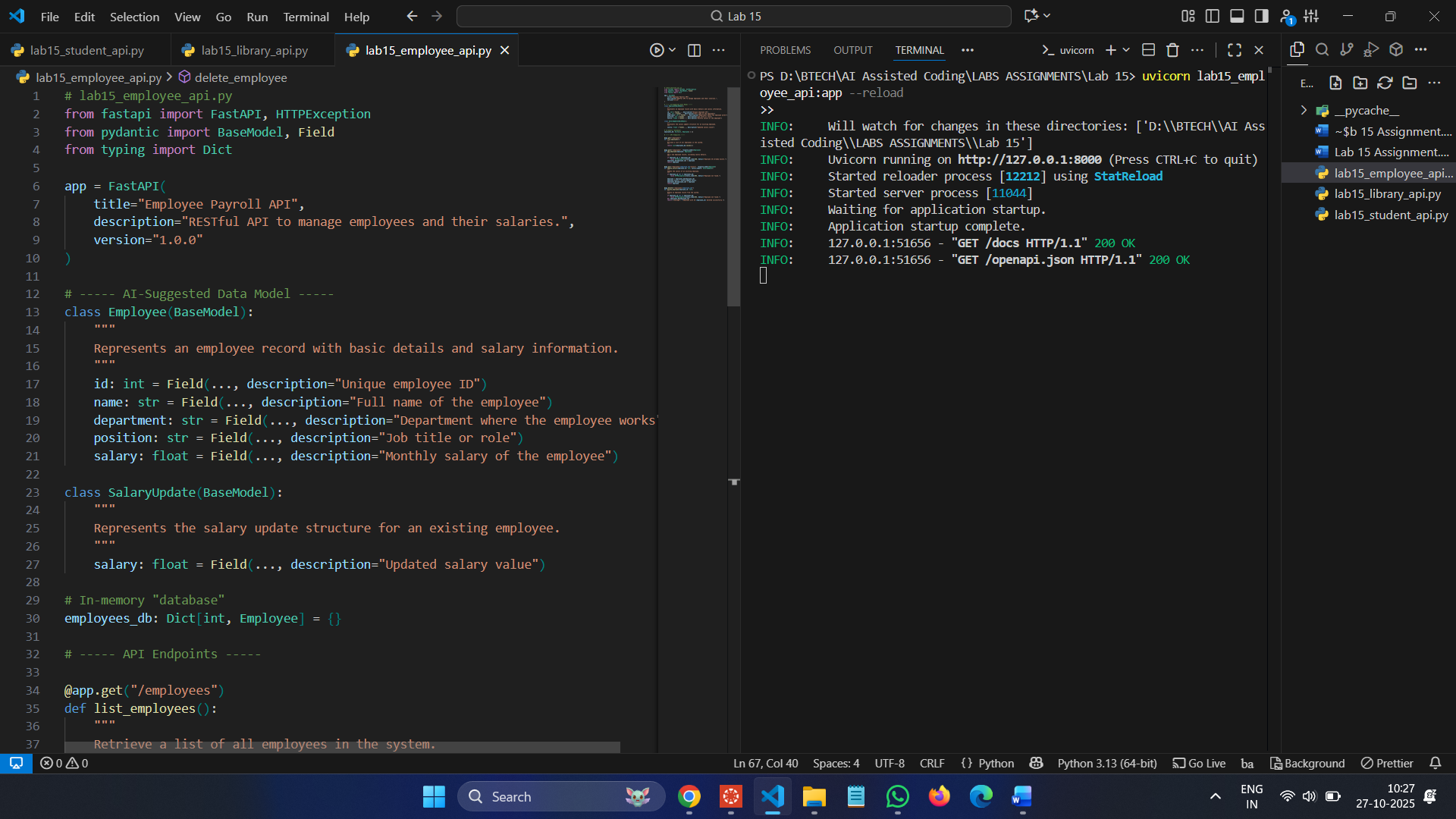


**DELETE :**



**Task 3 – Employee Payroll API  
Task:**  
Create an API for managing employees and their salaries.  
**Instructions:**  
• Endpoints required:  
o GET /employees → List all employees  
o POST /employees → Add a new employee with salary

details  
o PUT /employees/{id}/salary → Update salary of an  
employee  
o DELETE /employees/{id} → Remove employee from  
system  
**• Use AI to:**  
o Suggest data model structure.  
o Add comments/docstrings for all endpoints.  
**Code :**

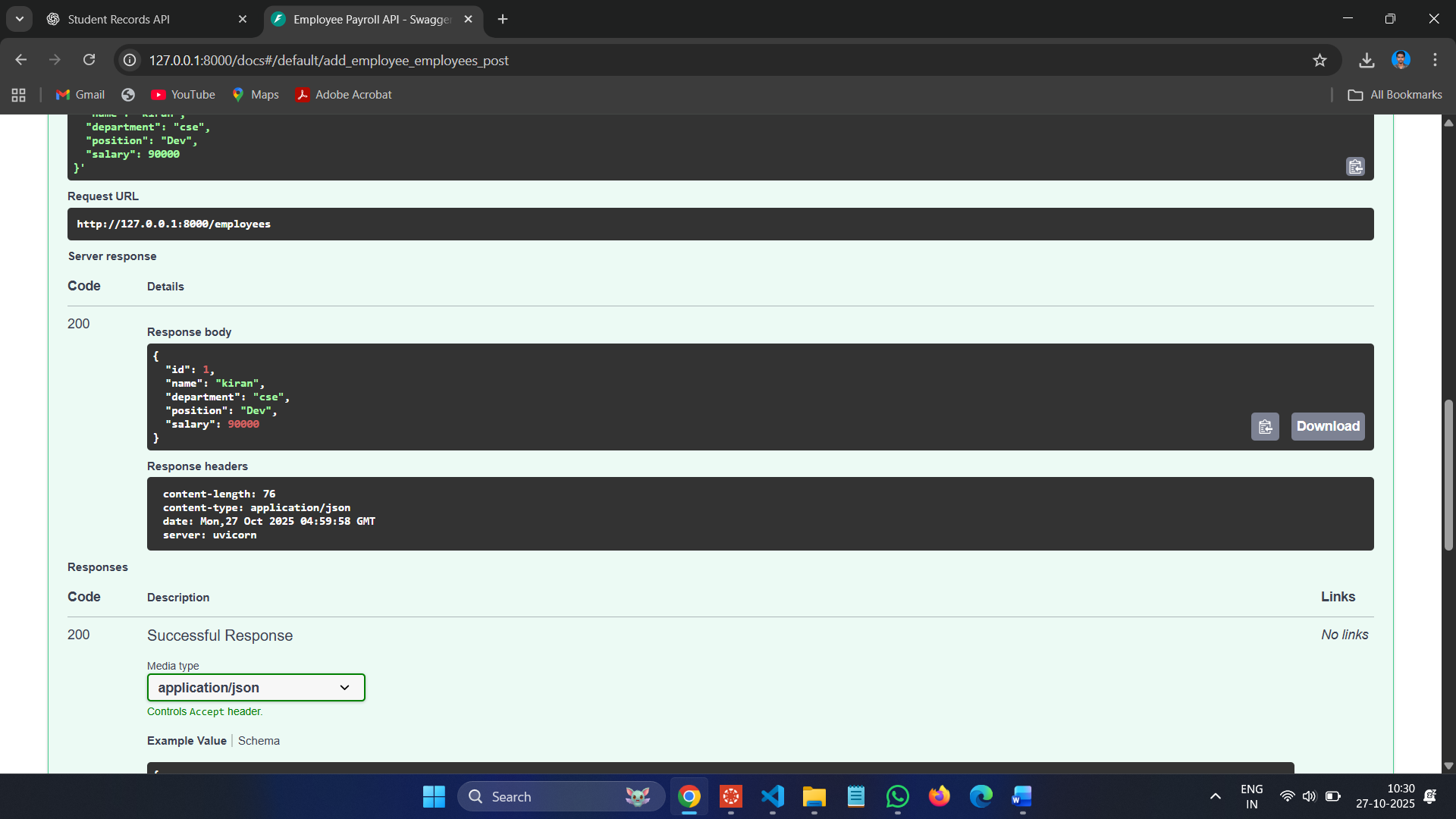


**Commands :**

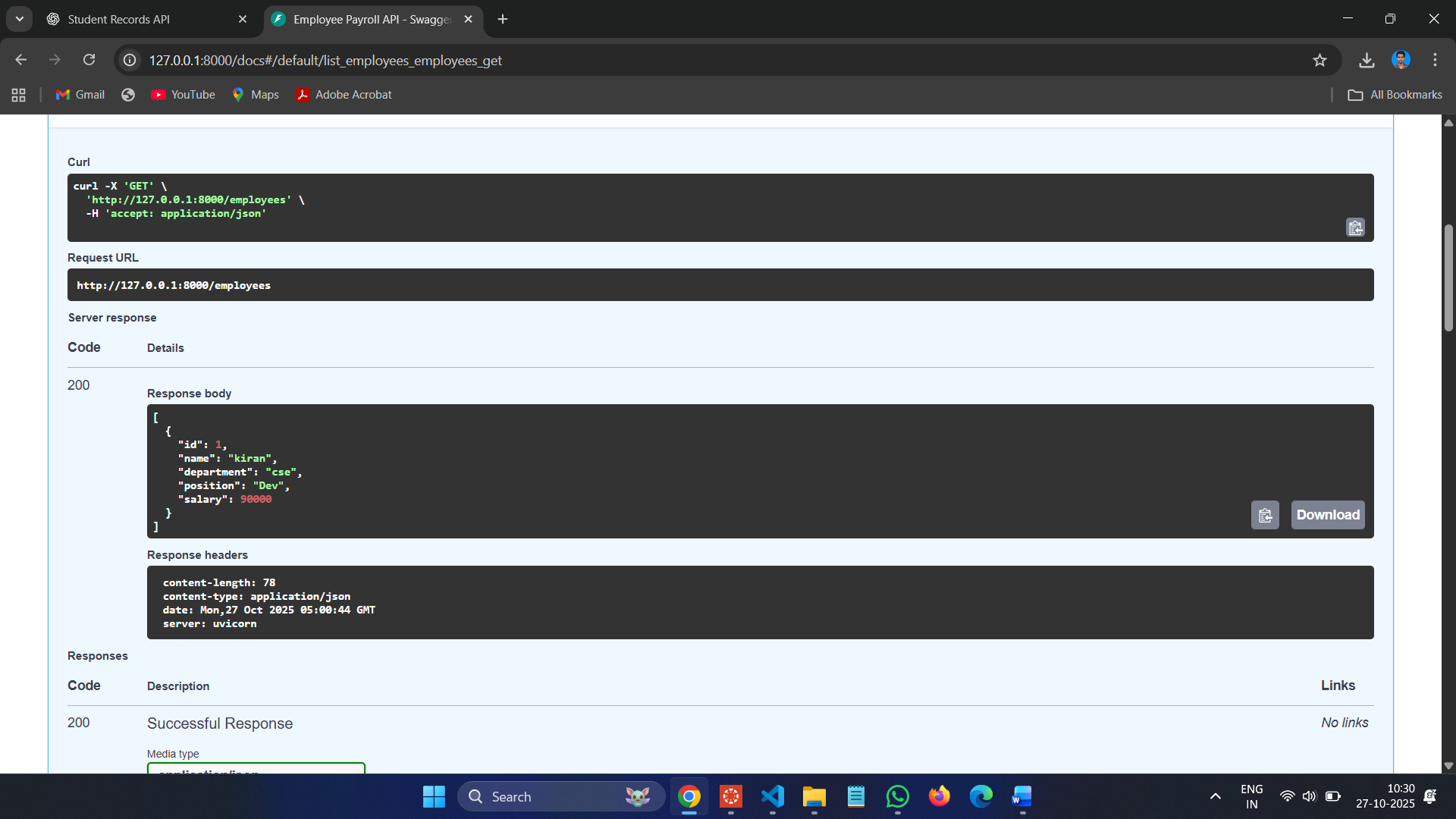
**uvicorn lab15\_employee\_api:app –reload**

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

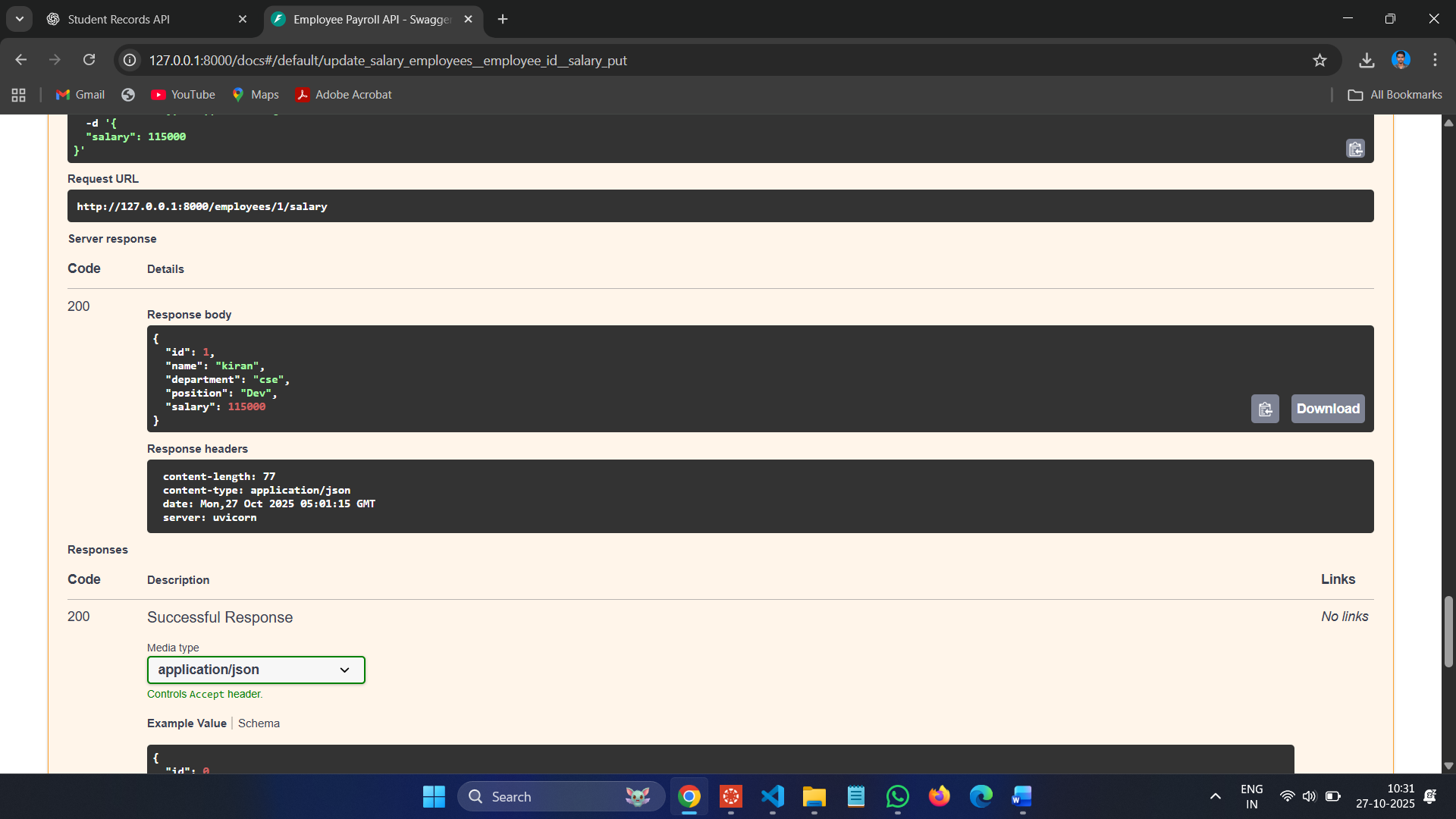
**POST :**



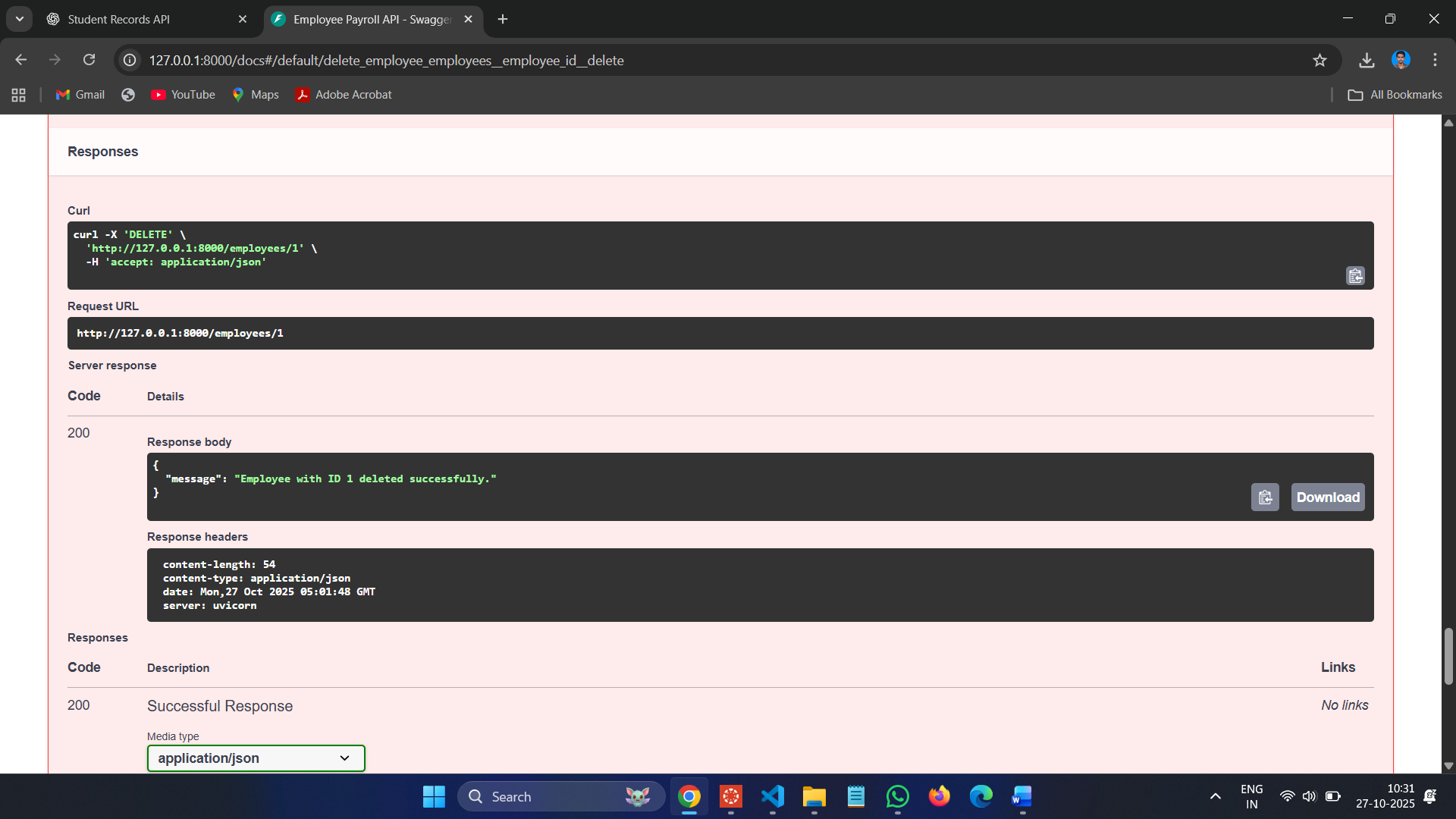
**GET :**



**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:**  
o 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:**  
o REST API code  
o Suggested improvements (like authentication, pagination)

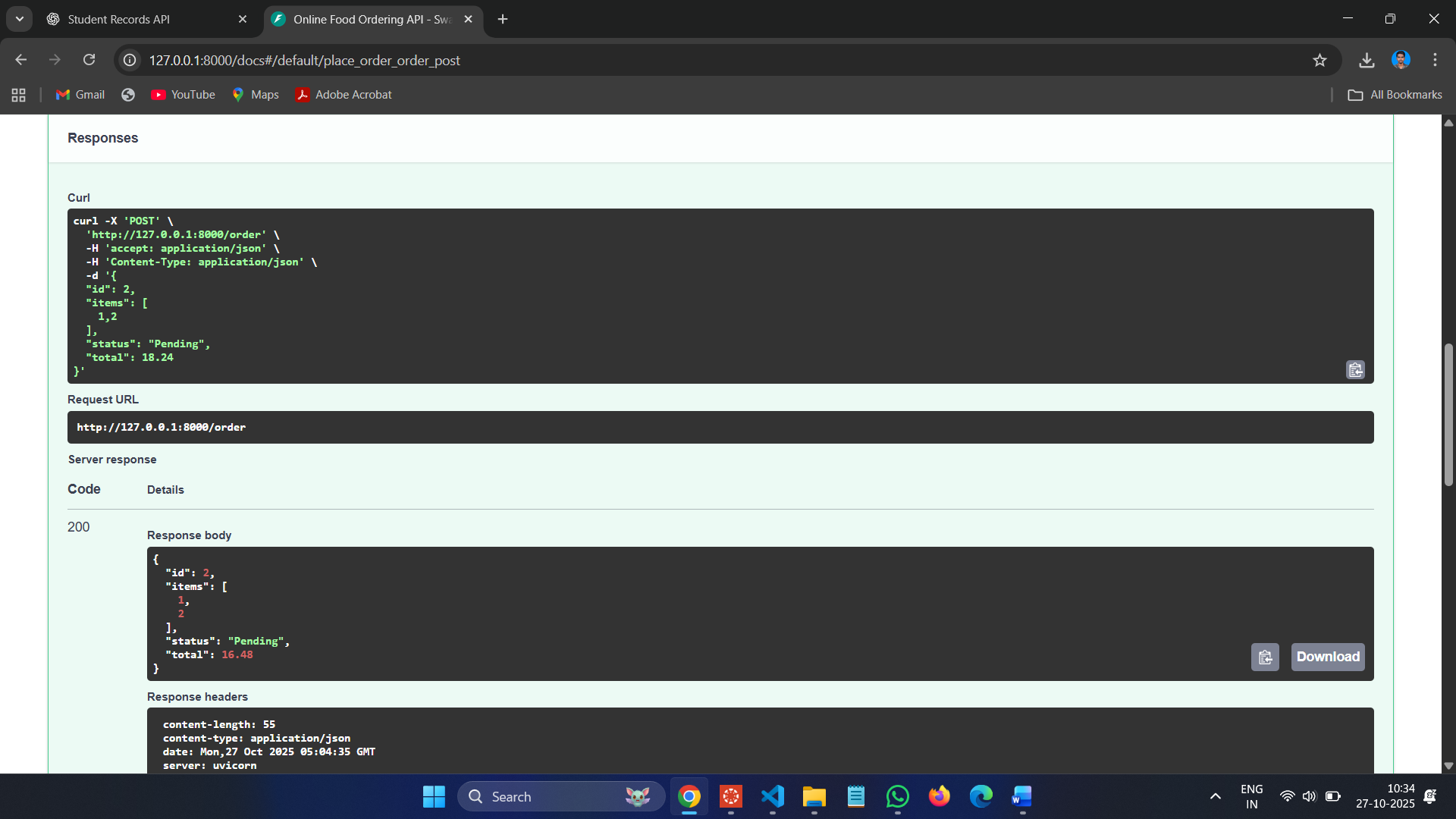
**Code :**



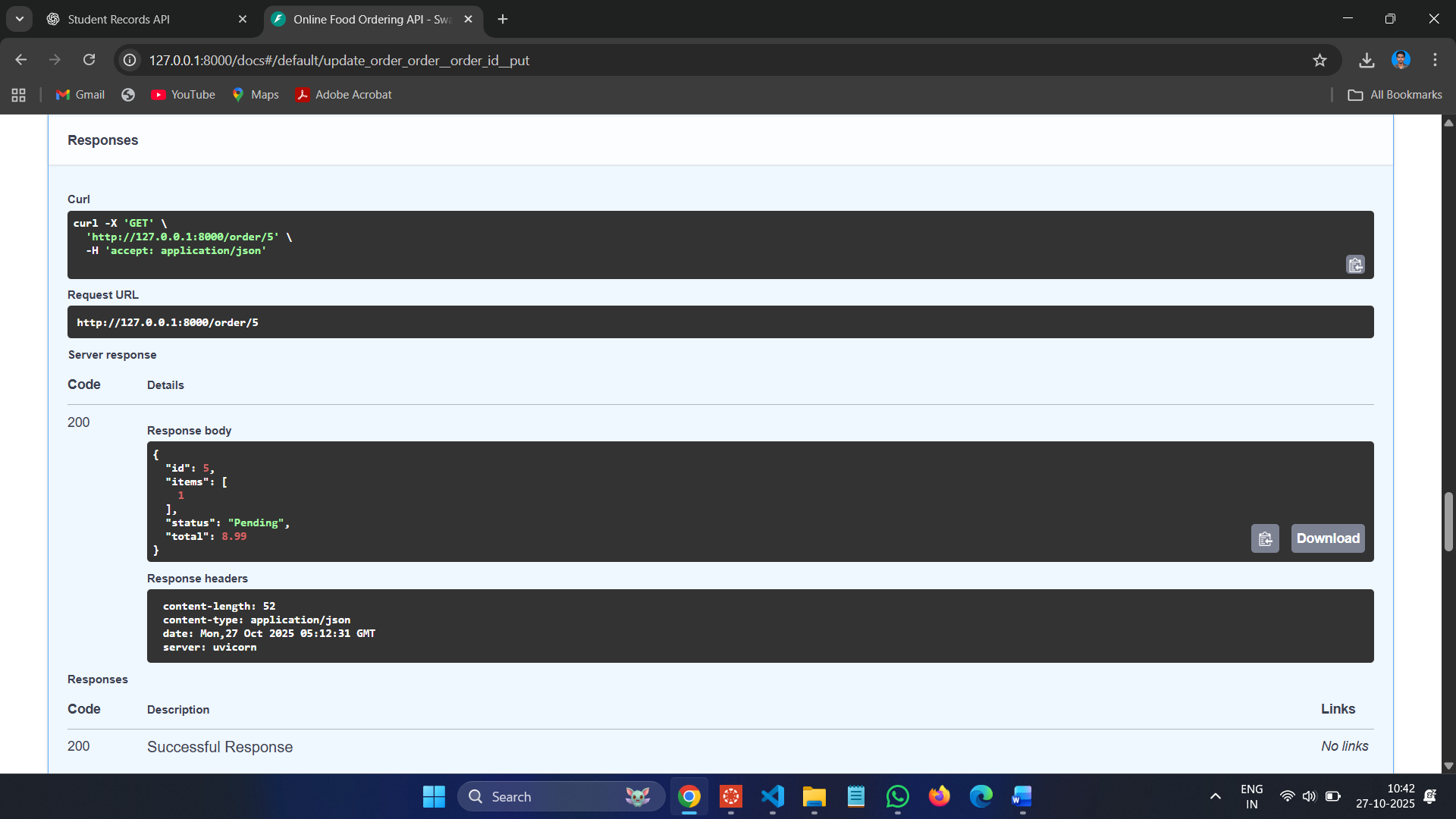
**Commands :**

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

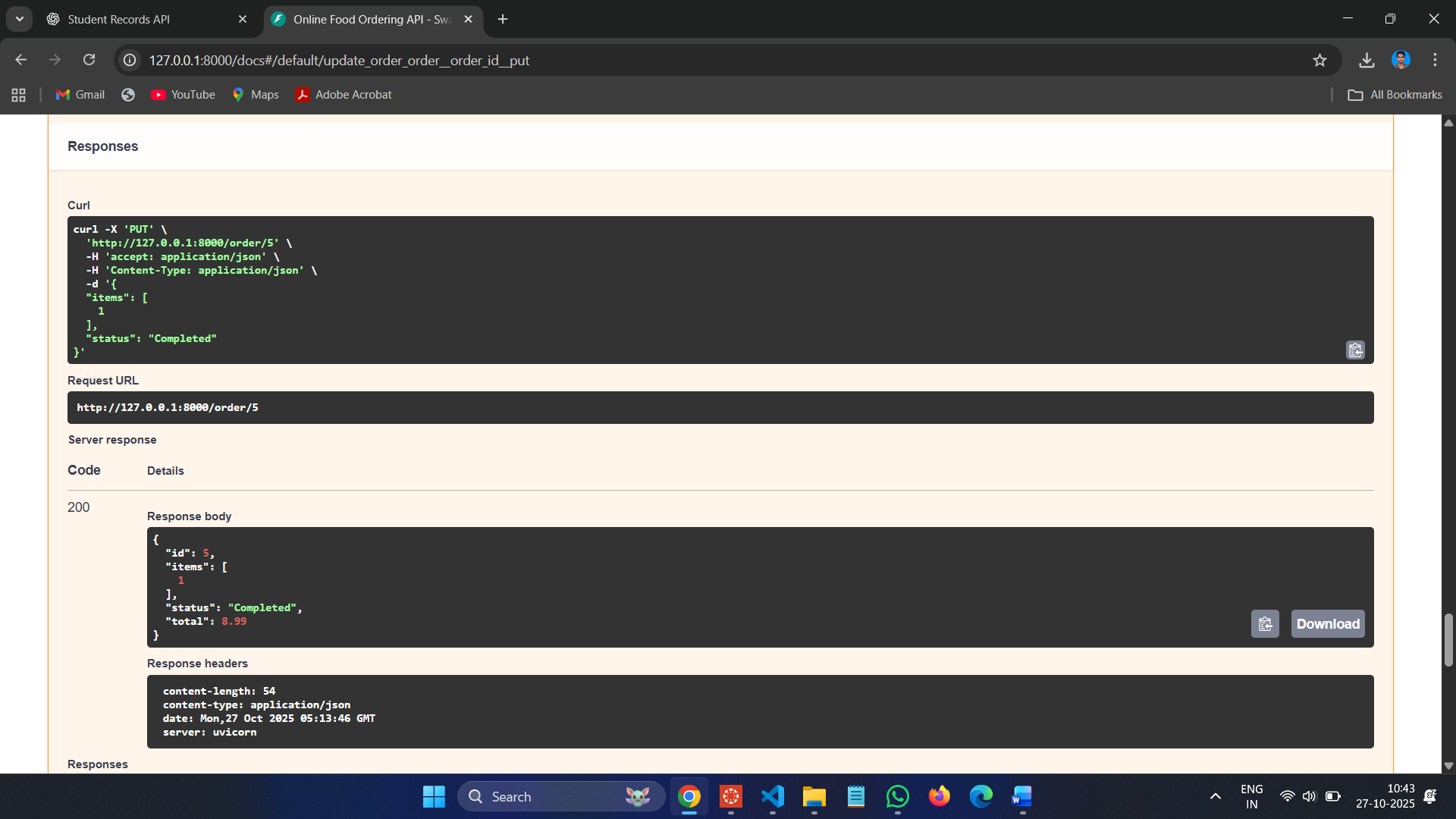
**Swagger UI:** [**http://127.0.0.1:8000/docs**](http://127.0.0.1:8000/docs)

**POST :**  


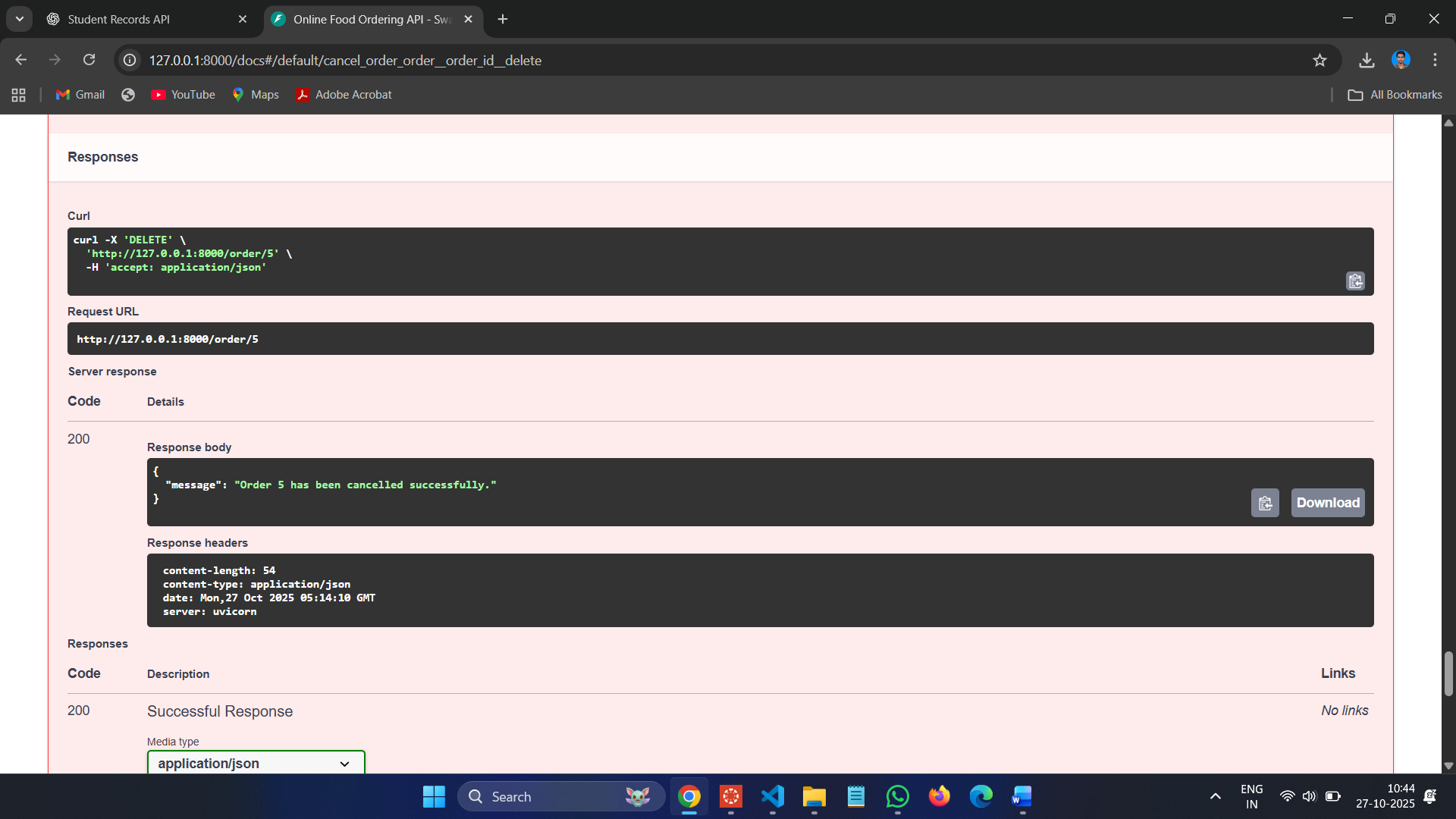
**GET :**



**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.
2. **AI-Assisted Code Generation:**  
   AI tools were effectively used to generate data models, endpoint structures, and documentation automatically, saving development time and ensuring consistency.
3. **CRUD Operations:**  
   All four APIs successfully implemented CRUD functionality:
   * Create: POST requests
   * Read: GET requests
   * Update: PUT/PATCH requests
   * Delete: DELETE requests
4. **Error Handling:**  
   Each API handled invalid requests gracefully using HTTPException with proper status codes and messages.
5. **Auto Documentation:**  
   FastAPI automatically provided API documentation through /docs (Swagger UI) and /redoc (ReDoc), fulfilling the lab’s documentation objective.
6. **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.
7. **AI-Driven Improvements:**  
   Suggested features such as authentication, pagination, real-time updates (via WebSockets), and database integration demonstrate how AI can enhance backend architecture planning.