

Assignment-15

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

NAME: R.Shivani

Htno : 2403A52411

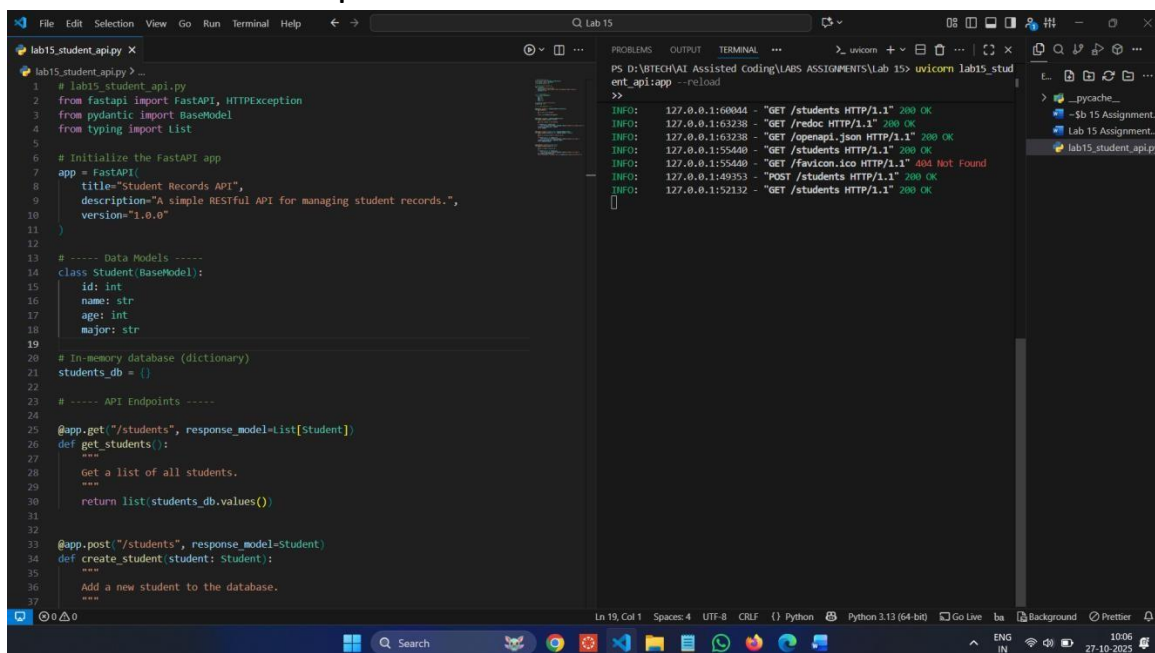
BATCH:15

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 :**



The screenshot displays a VS Code editor with a Python FastAPI application named `lab15_student_api.py`. The code defines a `Student` model, an in-memory database `students_db`, and three API endpoints: `GET /students` (listing all students), `POST /students` (adding a new student), and `PUT /students/{id}` (updating a student). The terminal shows the application running on `uvicorn` with several test requests and responses, including a 404 Not Found for a non-existent student ID.

```
lab15_student_api.py X
lab15_student_api.py
1 # lab15_student_api.py
2 from fastapi import FastAPI, HTTPException
3 from pydantic import BaseModel
4 from typing import List
5
6 # Initialize the FastAPI app
7 app = FastAPI(
8     title="Student Records API",
9     description="A simple RESTful API for managing student records.",
10    version="1.0.0"
11)
12
13 # ----- Data Models -----
14 class Student(BaseModel):
15     id: int
16     name: str
17     age: int
18     major: str
19
20 # In-memory database (dictionary)
21 students_db = {}
22
23 # ----- API Endpoints -----
24
25 @app.get("/students", response_model=List[Student])
26 def get_students():
27     """
28     Get a list of all students.
29     """
30     return list(students_db.values())
31
32
33 @app.post("/students", response_model=Student)
34 def create_student(student: Student):
35     """
36     Add a new student to the database.
37     """
38     # ... (code to add student to database) ...
39
40
41 @app.put("/students/{id}")
42 def update_student(id: int, student: Student):
43     """
44     Update student details.
45     """
46     # ... (code to update student) ...
47
48
49 @app.delete("/students/{id}")
50 def delete_student(id: int):
51     """
52     Delete a student record.
53     """
54     # ... (code to delete student) ...
55
56
57 if __name__ == "__main__":
58     uvicorn.run(app, host="0.0.0.0", port=8000)
```

Terminal Output:

```
PS D:\BTECH\AI Assisted Coding\LABS ASSIGNMENTS\Lab 15> uvicorn lab15_stud
ent_api:app --reload
>>
INFO: 127.0.0.1:60044 - "GET /students HTTP/1.1" 200 OK
INFO: 127.0.0.1:63238 - "GET /redoc HTTP/1.1" 200 OK
INFO: 127.0.0.1:63238 - "GET /openapi.json HTTP/1.1" 200 OK
INFO: 127.0.0.1:55440 - "GET /students HTTP/1.1" 200 OK
INFO: 127.0.0.1:55440 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:49353 - "POST /students HTTP/1.1" 200 OK
INFO: 127.0.0.1:52132 - "GET /students HTTP/1.1" 200 OK
```

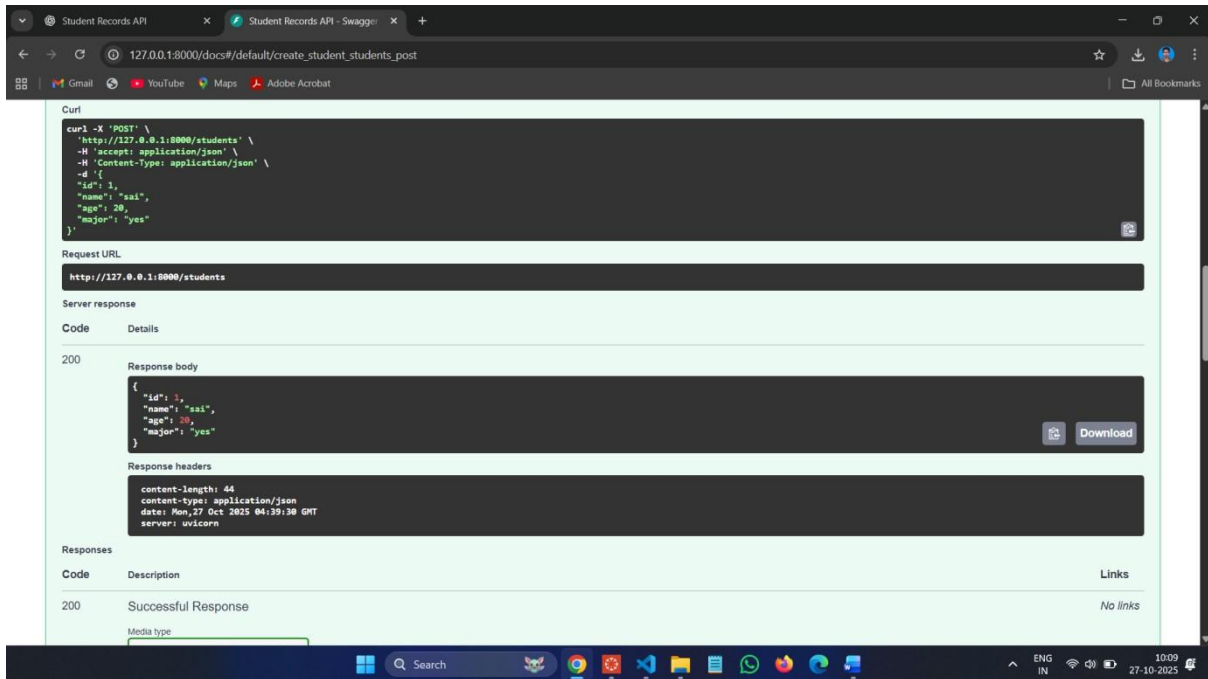
Swagger UI: <http://127.0.0.1:8000/docs>

Commands :

pip install fastapiuvicorn

uvicorn lab15_student_api:app --reload **Output:**

POST :



GET :

Student Records API - Swagger

127.0.0.1:8000/docs#/default/get_students_students_get

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/students' \
  -H 'accept: application/json'
```

Request URL

http://127.0.0.1:8000/students

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 1, "name": "sai", "age": 20, "major": "cse" }, { "id": 1, "name": "sai", "age": 20, "major": "yes" }</pre> <p>Response headers</p> <pre>content-length: 91 content-type: application/json date: Mon, 27 Oct 2025 04:48:34 GMT server: uvicorn</pre>

Responses

Code	Description	Links
200	Successful Response	No links

PUT :

Student Records API - Swagger

127.0.0.1:8000/docs#/default/update_student_students_student_id_put

Curl

```
curl -X 'PUT' \
  'http://127.0.0.1:8000/students/2' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 0,
    "name": "ram",
    "age": 0,
    "major": "string"
  }'
```

Request URL

http://127.0.0.1:8000/students/2

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 0, "name": "ram", "age": 0, "major": "string" }</pre> <p>Response headers</p> <pre>content-length: 46 content-type: application/json date: Mon, 27 Oct 2025 04:41:18 GMT server: uvicorn</pre>

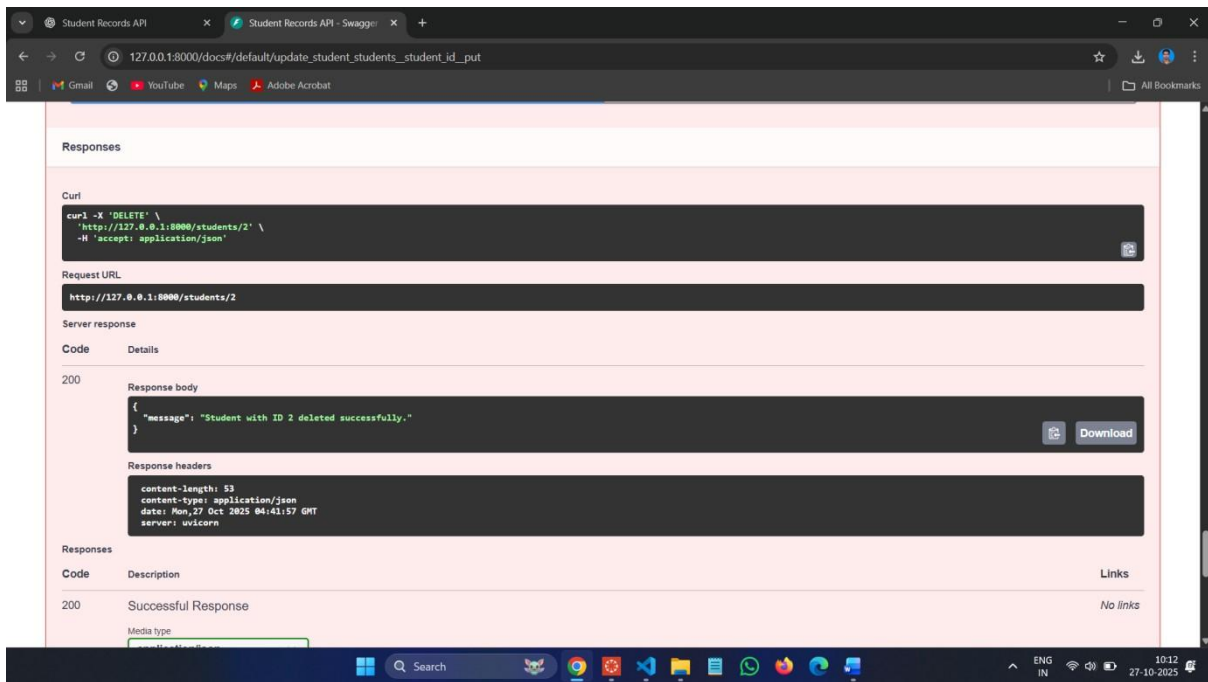
Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

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 :

The image shows a VS Code editor with two files open: `lab15_library_api.py` and `lab15_student_api.py`. The `lab15_library_api.py` file contains the following code:

```
1 # lab15_library_api.py
2 from fastapi import FastAPI, HTTPException
3 from pydantic import BaseModel
4 from typing import Optional, Dict
5
6
7 app = FastAPI(
8     title="Library Book Management API",
9     description="RESTful API to manage library books with CRUD and partial update",
10     version="1.0.0"
11 )
12
13 # ----- Data Models -----
14 class Book(BaseModel):
15     id: int
16     title: str
17     author: str
18     year: int
19     available: bool = True
20
21 class BookUpdate(BaseModel):
22     title: Optional[str] = None
23     author: Optional[str] = None
24     year: Optional[int] = None
25     available: Optional[bool] = None
26
27 # In-memory "database"
28 books_db: Dict[int, Book] = {}
29
30 # ----- Endpoints -----
31
32 @app.get("/books")
33 def get_books():
34     """
35     Retrieve all books in the library.
36     """
37     return list(books_db.values())
```

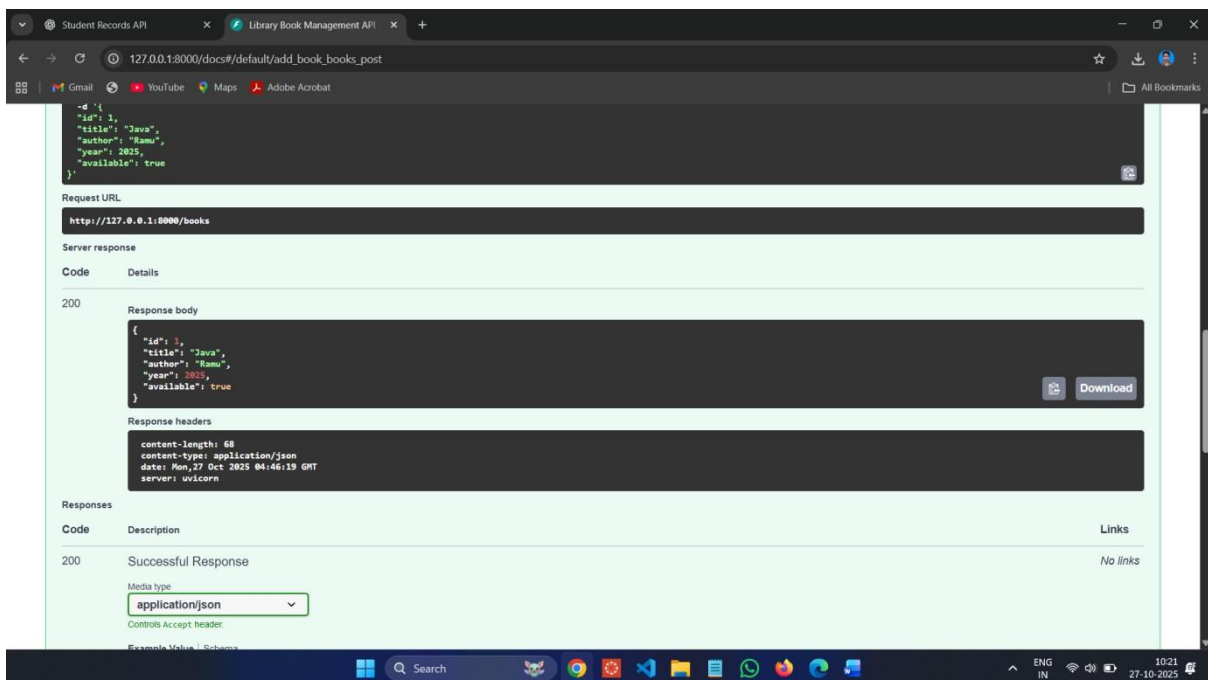
The terminal output shows the command `uvicorn lab15_library_api:app --reload` being executed. The output indicates that the application is running on `http://127.0.0.1:8000` and that the reload process is active. The terminal also shows the application startup logs, including the server process starting and the application startup complete.

Commands :

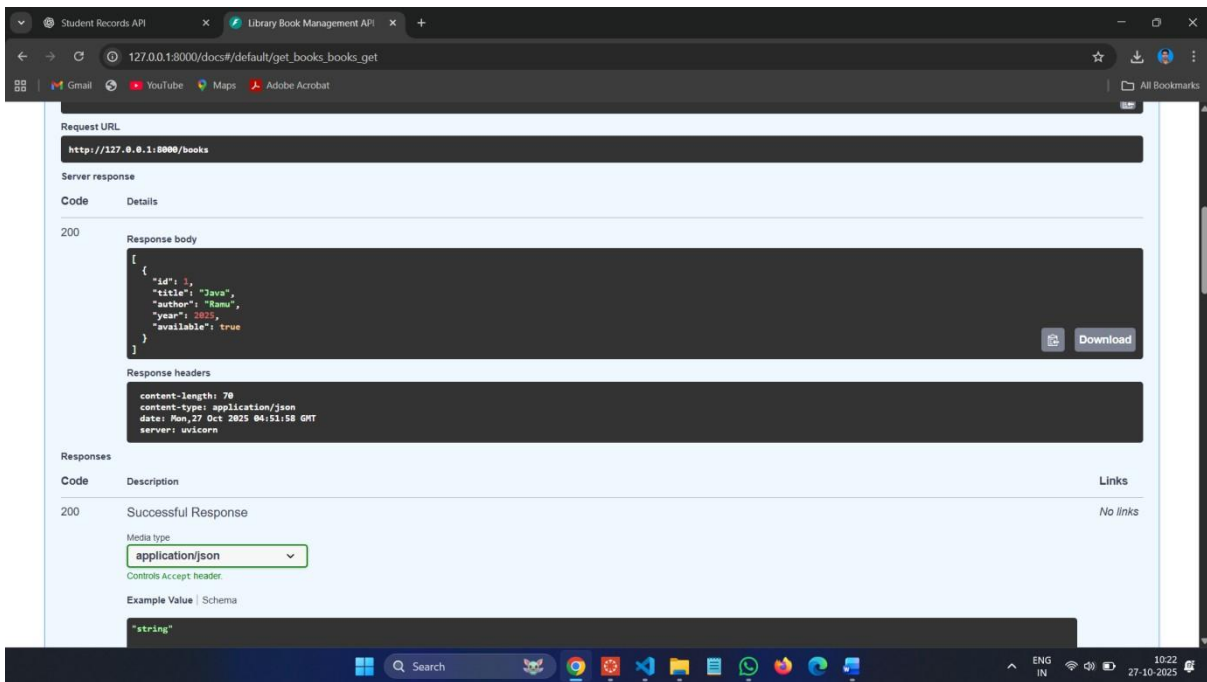
`uvicorn lab15_library_api:app --reload`

Swagger UI: <http://127.0.0.1:8000/docs>

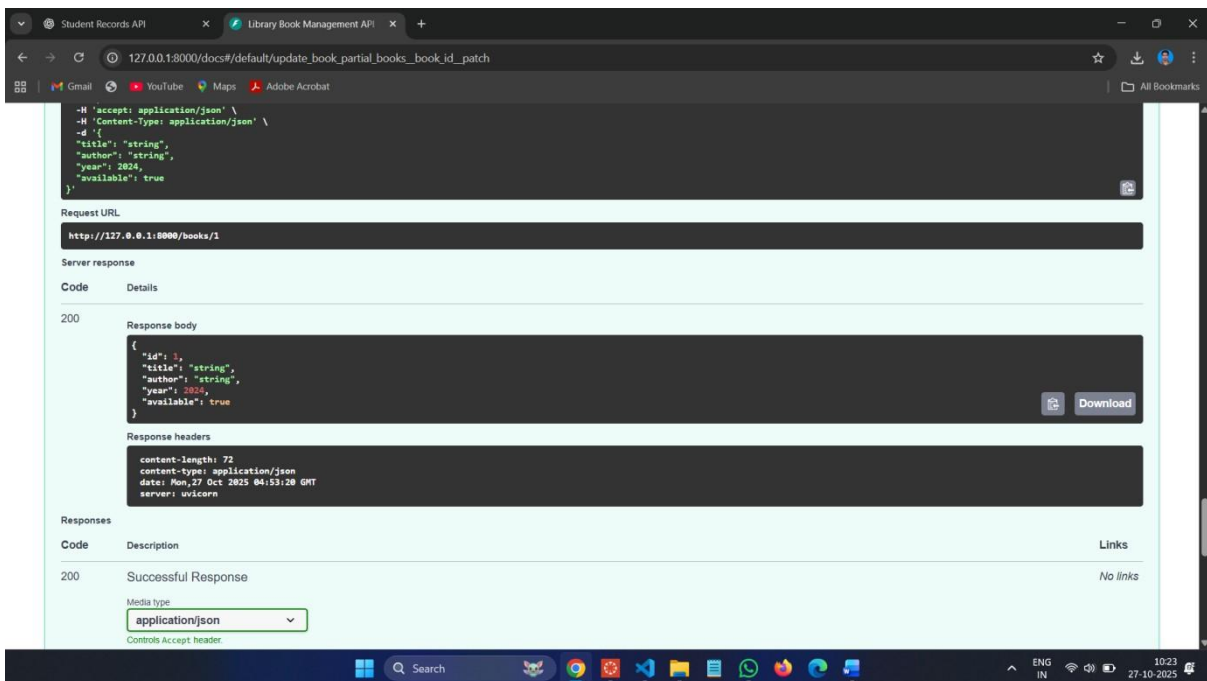
POST :



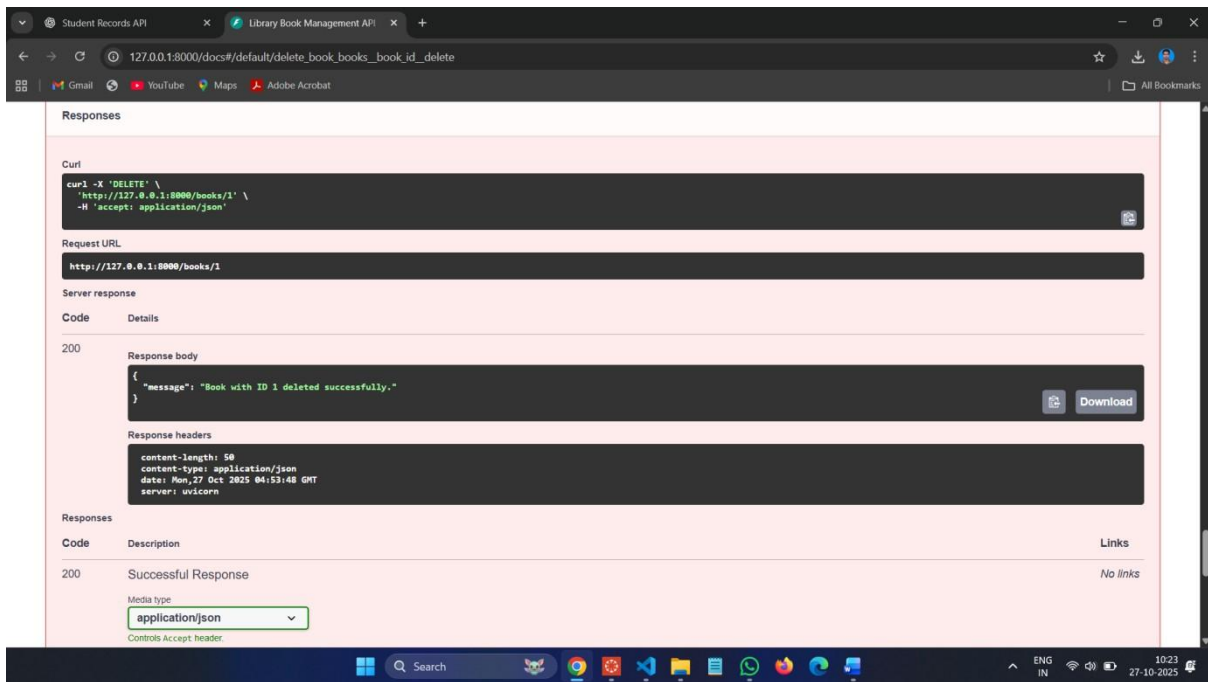
GET :



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 :**

The screenshot shows a VS Code editor with a Python FastAPI application. The code defines an `Employee` model and a `listEmployees` endpoint. The terminal shows the application running on `http://127.0.0.1:8000` with a reload command. The file explorer on the right shows the project structure.

```
lab15_employee_api.py > delete_employee
# lab15_employee_api.py
1 from fastapi import FastAPI, HTTPException
2 from pydantic import BaseModel, Field
3 from typing import Dict
4
5
6 app = FastAPI(
7     title="Employee Payroll API",
8     description="RESTful API to manage employees and their salaries.",
9     version="1.0.0"
10 )
11
12 # ----- AI-Suggested Data Model -----
13 class Employee(BaseModel):
14     """
15     Represents an employee record with basic details and salary information.
16     """
17     id: int = Field(..., description="Unique employee ID")
18     name: str = Field(..., description="Full name of the employee")
19     department: str = Field(..., description="Department where the employee works")
20     position: str = Field(..., description="Job title or role")
21     salary: float = Field(..., description="Monthly salary of the employee")
22
23 class SalaryUpdate(BaseModel):
24     """
25     Represents the salary update structure for an existing employee.
26     """
27     salary: float = Field(..., description="Updated salary value")
28
29 # In-memory "database"
30 employees_db: Dict[int, Employee] = {}
31
32 # ----- API Endpoints -----
33
34 @app.get("/employees")
35 def list_employees():
36     """
37     Retrieve a list of all employees in the system.
38     """
```

Terminal Output:

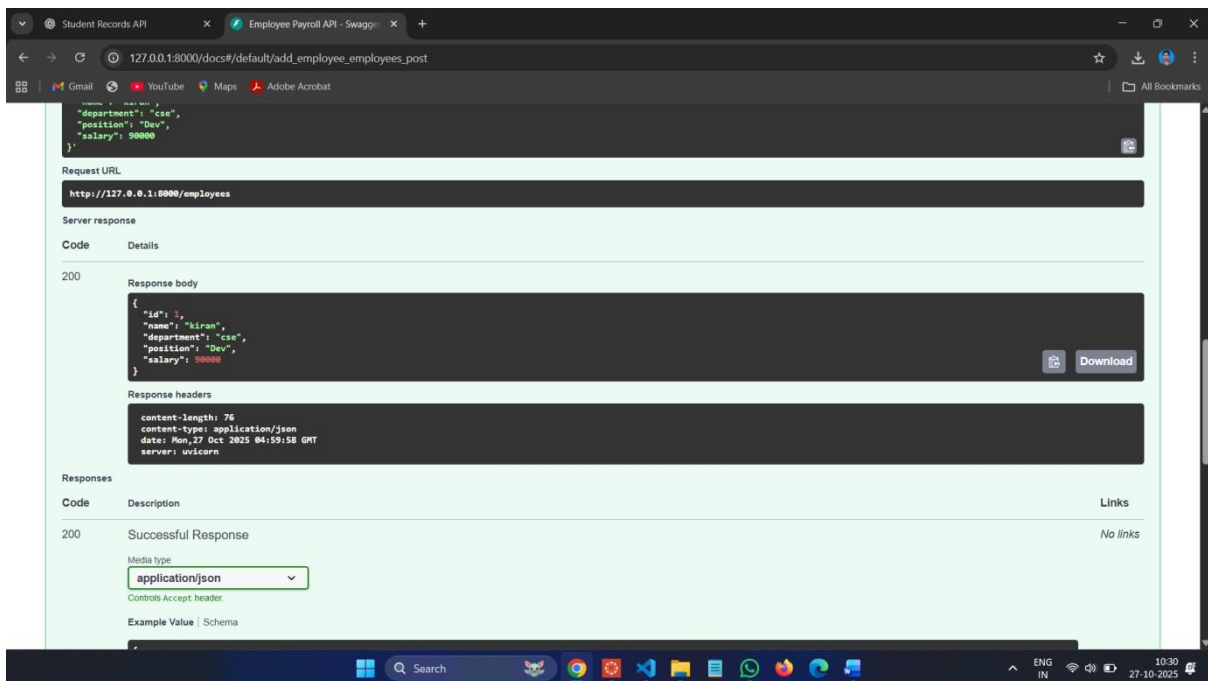
```
PS D:\VBTECH\AI Assisted Coding\LABS ASSIGNMENTS\Lab 15> uvicorn lab15_employee_api:app --reload
INFO: Will watch for changes in these directories: ['D:\VBTECH\AI Assisted Coding\LABS ASSIGNMENTS\Lab 15']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reload process [12212] using Stateload
INFO: Started server process [11004]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:51656 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:51656 - "GET /openapi.json HTTP/1.1" 200 OK
```

Commands :

uvicorn lab15_employee_api:app --reload

Swagger UI: <http://127.0.0.1:8000/docs> POST

:



GET :

Student Records API Employee Payroll API - Swagger

127.0.0.1:8000/docs/default/list_employees_employees_get

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/employees' \
  -H 'accept: application/json'
```

Request URL

http://127.0.0.1:8000/employees

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 1, "name": "kiran", "department": "cse", "position": "Dev", "salary": 90000 }</pre> <p>Response headers</p> <pre>content-length: 78 content-type: application/json date: Mon, 27 Oct 2025 05:00:44 GMT server: uvicorn</pre>

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

PUT :

Student Records API Employee Payroll API - Swagger

127.0.0.1:8000/docs/default/update_salary_employees_employee_id_salary_put

Request body

```
-d '{
  "salary": 115000
}'
```

Request URL

http://127.0.0.1:8000/employees/1/salary

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 1, "name": "kiran", "department": "cse", "position": "Dev", "salary": 115000 }</pre> <p>Response headers</p> <pre>content-length: 77 content-type: application/json date: Mon, 27 Oct 2025 05:01:15 GMT server: uvicorn</pre>

Responses

Code	Description	Links
200	Successful Response	No links

Media type

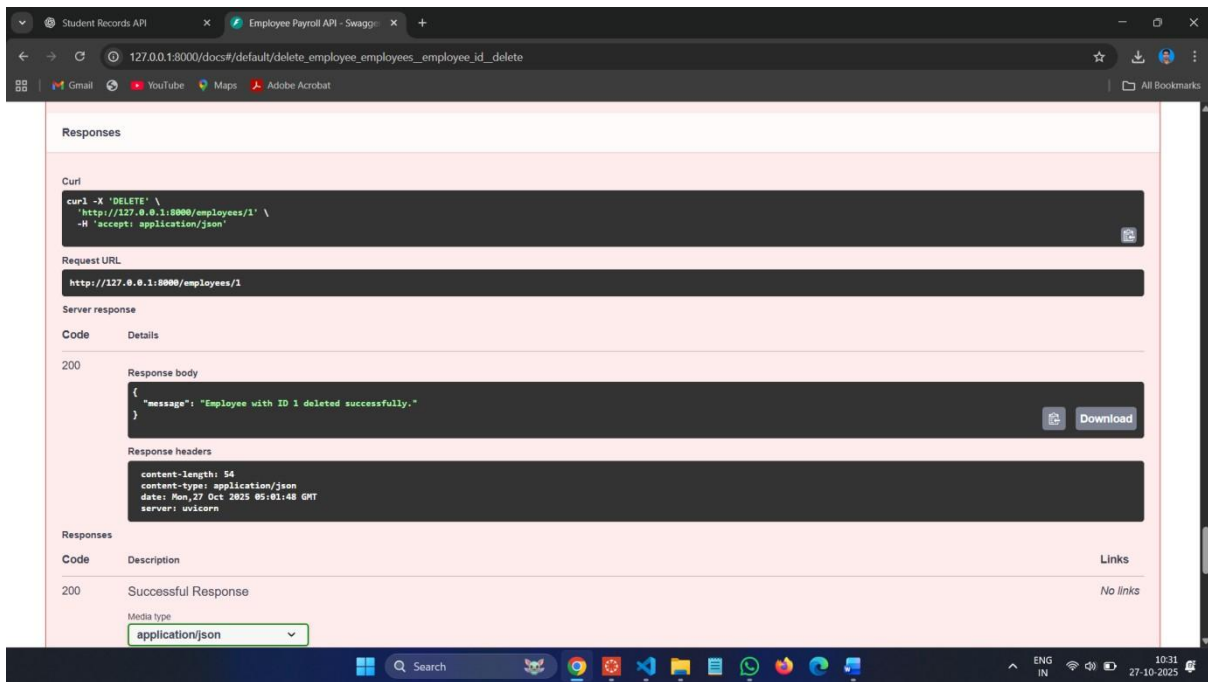
application/json

Controls Accept header

Example Value | Schema

```
{
  "id": 1
}
```

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 :

The image shows a VS Code editor with a Python FastAPI application named `lab15_food_api.py`. The code defines a RESTful API for an online food ordering system. It includes a `FastAPI` app, a `Dish` model, an `Order` model, and an `OrderUpdate` model. The `main` function initializes an in-memory database with four dishes: Margherita Pizza, Cheeseburger, Pasta Alfredo, and Caesar Salad. The terminal output shows the application running on `http://127.0.0.1:8000` and successfully handling three requests: `GET /docs HTTP/1.1`, `GET /openapi.json HTTP/1.1`, and `POST /order HTTP/1.1`.

```
1 # lab15_food_api.py
2 from fastapi import FastAPI, HTTPException
3 from pydantic import BaseModel, Field
4 from typing import List, Dict, Optional
5
6 app = FastAPI(
7     title="Online Food Ordering API",
8     description="A RESTful API that simulates a basic online food ordering system",
9     version="1.0.0"
10 )
11
12 # ----- Data Models -----
13 class Dish(BaseModel):
14     """Represents a menu item (dish) available for order."""
15     id: int
16     name: str
17     price: float
18
19 class Order(BaseModel):
20     """Represents a customer's food order."""
21     id: int
22     items: List[int] = Field(..., description="List of dish IDs included in the order")
23     status: str = Field(default="Pending", description="Current status of the order")
24     total: float = 0.0
25
26 class OrderUpdate(BaseModel):
27     """Represents updates that can be applied to an existing order."""
28     items: Optional[List[int]] = None
29     status: Optional[str] = None
30
31 # ----- In-memory databases -----
32 menu_db: Dict[int, Dish] = {
33     1: Dish(id=1, name="Margherita Pizza", price=8.99),
34     2: Dish(id=2, name="Cheeseburger", price=7.49),
35     3: Dish(id=3, name="Pasta Alfredo", price=9.25),
36     4: Dish(id=4, name="Caesar Salad", price=5.75),
37 }
```

Terminal Output:

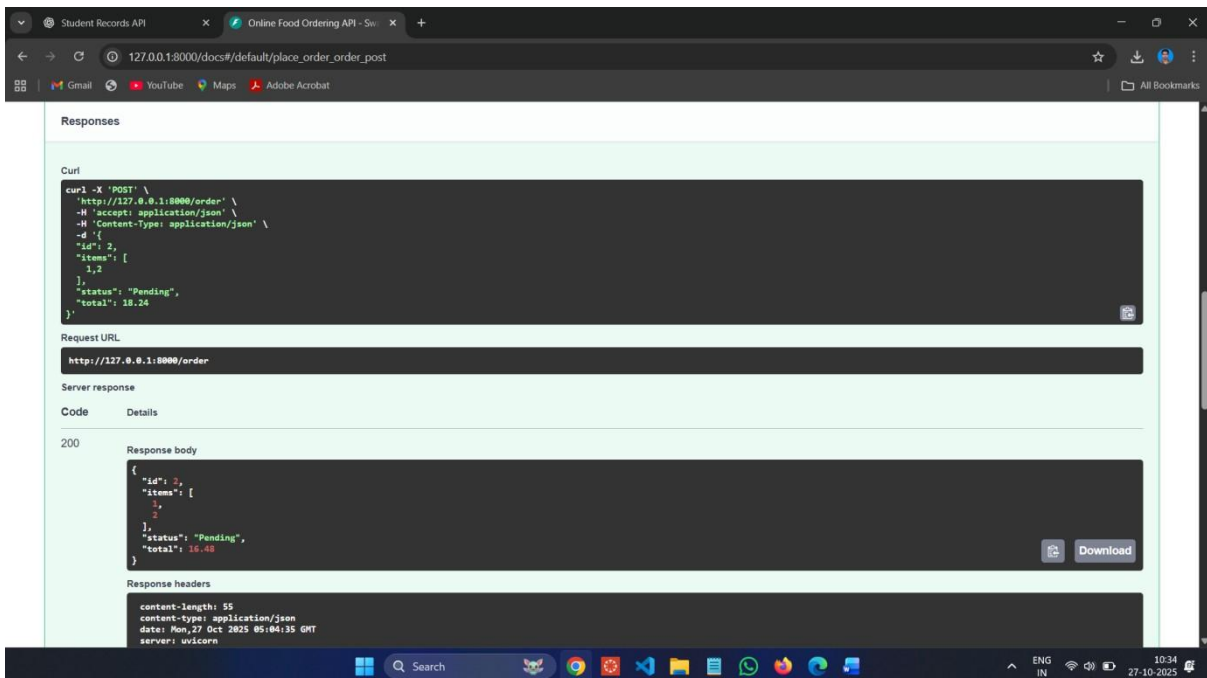
```
PS D:\VBTECH\AI Assisted Coding\LABS ASSIGNMENTS\Lab 15> uvicorn lab15_food_api:app --reload
INFO: Will watch for changes in these directories: ['D:\VBTECH\AI Assisted Coding\LABS ASSIGNMENTS\Lab 15']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reload process [3408] using Statreload
INFO: Started server process [1936]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:53575 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:53575 - "GET /openapi.json HTTP/1.1" 200 OK
INFO: 127.0.0.1:63392 - "POST /order HTTP/1.1" 200 OK
```

Commands :

`uvicorn lab15_food_api:app --reload`

Swagger UI: <http://127.0.0.1:8000/docs>

POST :



GET :

Student Records API Online Food Ordering API - Sw

127.0.0.1:8000/docs/default/update_order_order_id_put

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/order/5' \
  -H 'accept: application/json'
```

Request URL

http://127.0.0.1:8000/order/5

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 5, "items": [1], "status": "Pending", "total": 8.99 }</pre> <p>Response headers</p> <pre>content-length: 52 content-type: application/json date: Mon, 27 Oct 2025 05:12:31 GMT server: uvicorn</pre>

Responses

Code	Description	Links
200	Successful Response	No links

PUT :

Student Records API Online Food Ordering API - Sw

127.0.0.1:8000/docs/default/update_order_order_id_put

Responses

Curl

```
curl -X 'PUT' \
  'http://127.0.0.1:8000/order/5' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "items": [
      1
    ],
    "status": "Completed"
  }'
```

Request URL

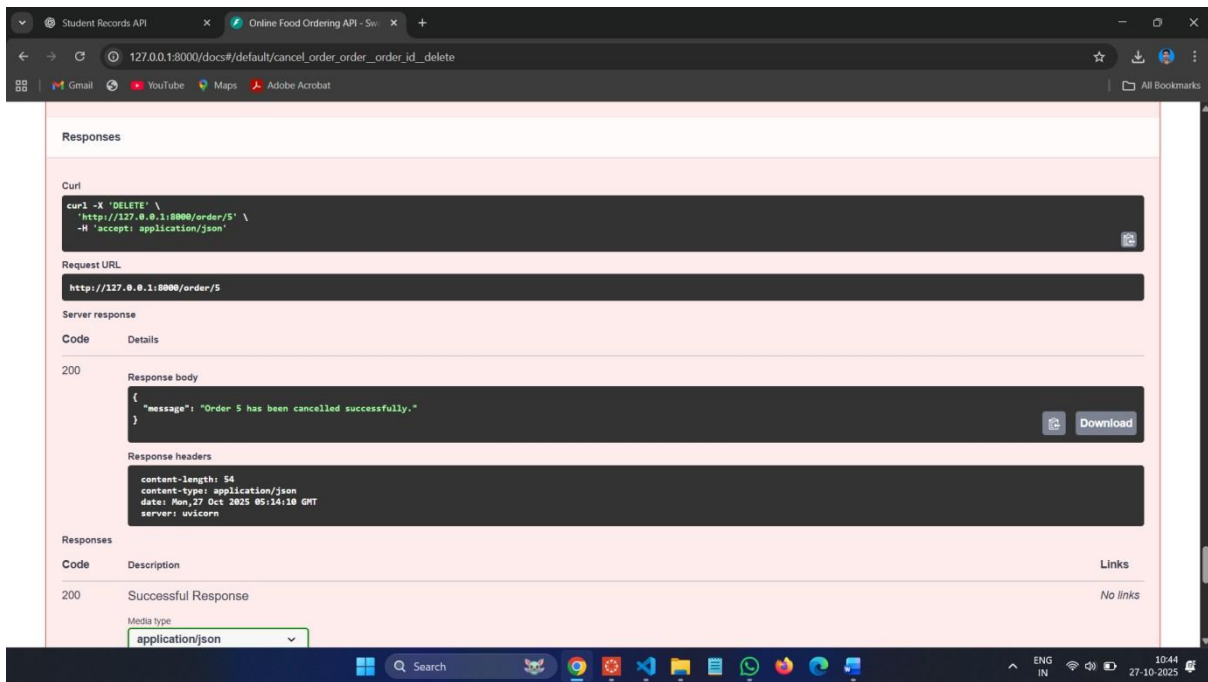
http://127.0.0.1:8000/order/5

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 5, "items": [1], "status": "Completed", "total": 8.99 }</pre> <p>Response headers</p> <pre>content-length: 54 content-type: application/json date: Mon, 27 Oct 2025 05:13:46 GMT server: uvicorn</pre>

Responses

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:

o Create: POST requests o Read: GET requests o Update:
PUT/PATCH requests o 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:

- o Task 2 introduced partial updates (PATCH) for library books.
- o 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, realtime updates (via WebSockets), and database integration demonstrate how AI can enhance backend architecture planning.