# ASSIGNMENT-15.1

NAME: VITTAM VENKATESH.

HALLTICKET NUNMER: 2403A52419

BATCH:15

SUBJECT:AI CODING

## TASK-1

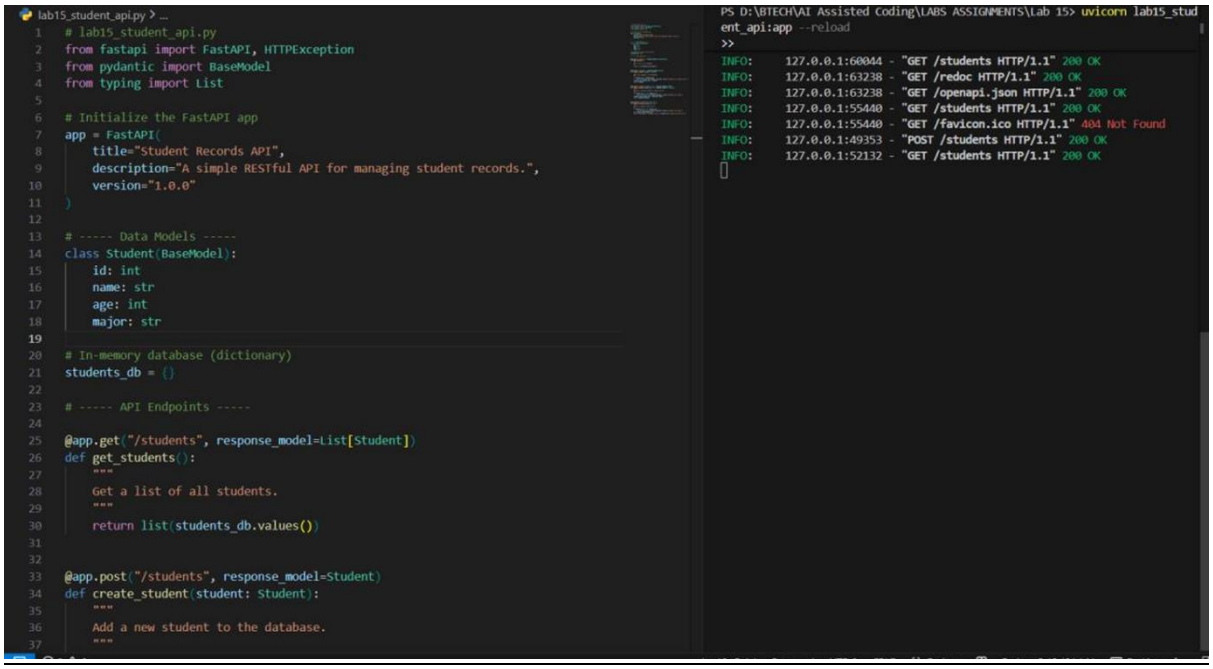
**QUESTION:**

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.
Expected Output:
• Working API with CRUD functionality for student records

**PROMPT:**

Create a simple python FastAPI CRUD API for student records using an in-memory list. Include endpoints for Get, Post, Put, Delete, and return json responses.

**CODE:**



```python
# lab15_student_api.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List

# Initialize the FastAPI app
app = FastAPI(
    title="Student Records API",
    description="A simple RESTful API for managing student records.",
    version="1.0.0"
)

# ----- Data Models -----
class Student(BaseModel):
    id: int
    name: str
    age: int
    major: str

# In-memory database (dictionary)
students_db = {}

# ----- API Endpoints -----

@app.get("/students", response_model=List[Student])
def get_students():
    """
    Get a list of all students.
    """
    return list(students_db.values())


@app.post("/students", response_model=Student)
def create_student(student: Student):
    """
    Add a new student to the database.
    """
```

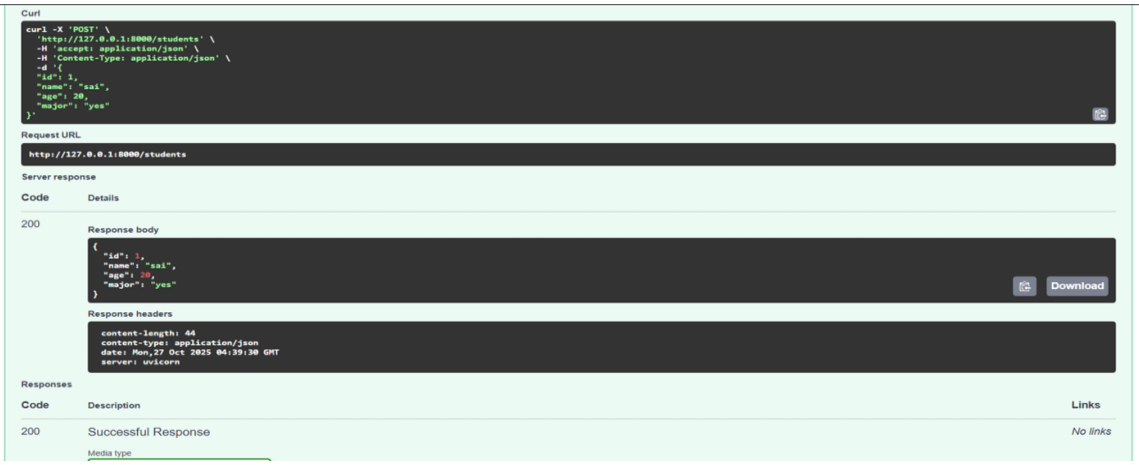
```
PS D:\BTECH\AI Assisted Coding\LABS ASSIGNMENTS\Lab 15> uvicorn lab15_stud
ent_api:app --reload
>>
INFO:     127.0.0.1:60044 - "GET /students HTTP/1.1" 200 OK
INFO:     127.0.0.1:63238 - "GET /redoc HTTP/1.1" 200 OK
INFO:     127.0.0.1:63238 - "GET /openapi.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:55440 - "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
```

**Commands : pip install fastapiuvicorn**

**uvicorn lab15_student_api:app –reload**

**OUTPUT:**

**Post:**



Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/students' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": 1,
  "name": "sai",
  "age": 20,
  "major": "yes"
}'
```

Request URL

```
http://127.0.0.1:8000/students
```

Server response

| Code | Details |
|---|---|
| 200 | Response body |

```
{
  "id": 1,
  "name": "sai",
  "age": 20,
  "major": "yes"
}
```

Download

Response headers

```
content-length: 44
content-type: application/json
date: Mon,27 Oct 2025 04:39:30 GMT
server: uvicorn
```

Responses

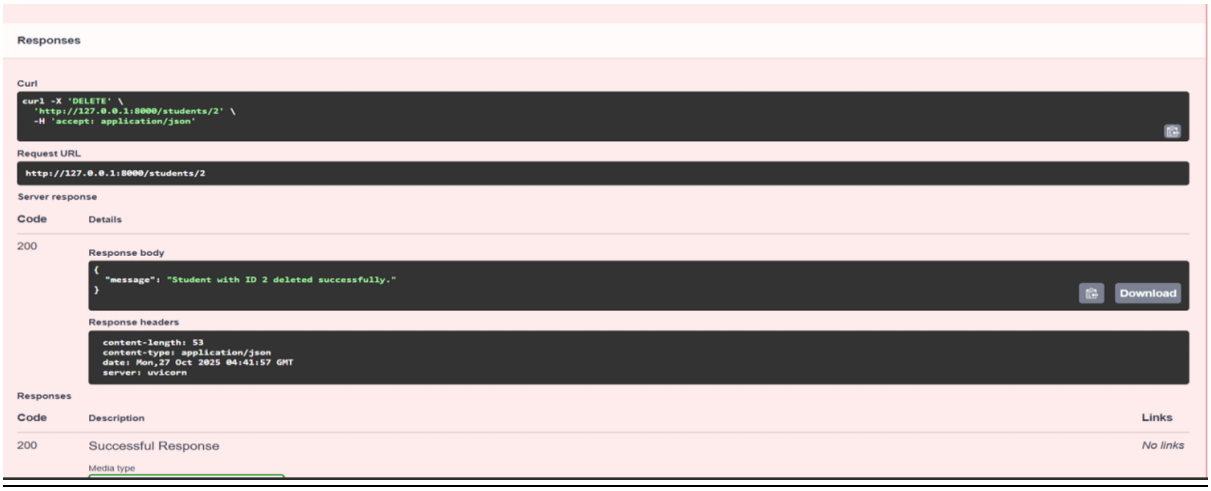
| Code | Description | Links |
|---|---|---|
| 200 | Successful Response | *No links* |

Media type

## Get:


## Put:


## Delete:



**OBSERVATION:**

The student records api successfully demonstrates basic crud operations using fastapi. it allows users to add, view, update, and delete student details through simple endpoints. the use of an in-memory list makes the implementation easy to understand and ideal for beginners learning restful apis. the api responses are returned in json format, ensuring clear and structured communication between the client and server. overall, it effectively shows how to manage data in a clean and user-friendly way.

# TASK-2

**QUESTION:**

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.
Expected Output:
• Functional API with CRUD + partial updates

**PROMPT:**

Create a fastapi based restful api for library book management using an in memory list. include endpoints for get, post, get by id, patch, and delete. handle invalid requests with proper error messages and return all responses in json format

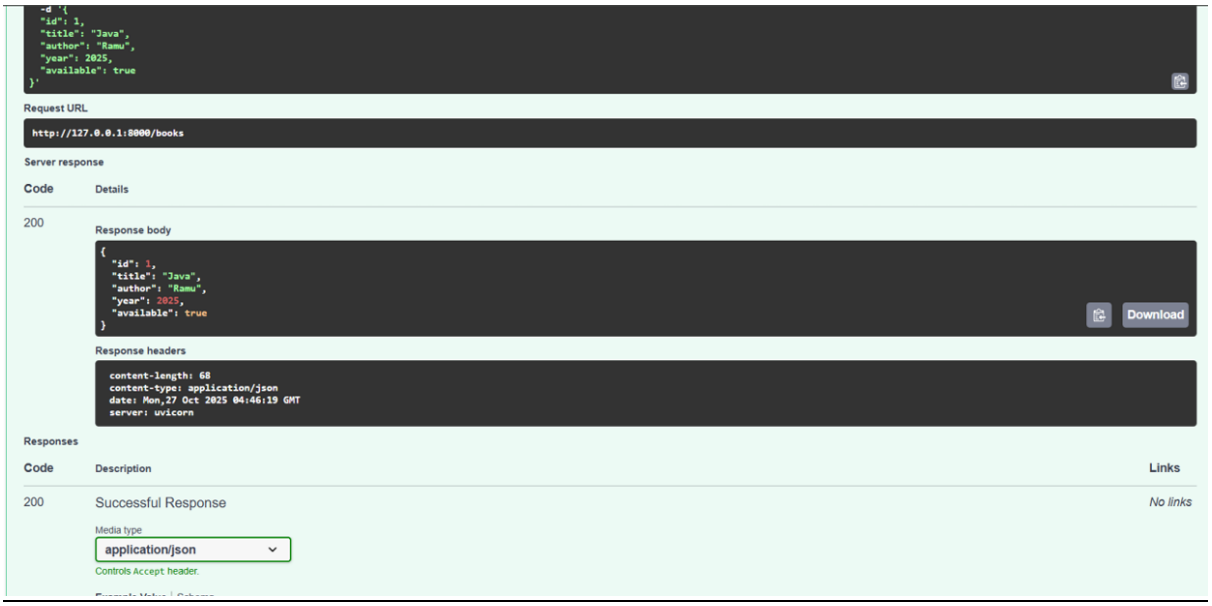
**CODE:**

```python
# lab15_library_api.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import Optional, Dict

app = FastAPI(
    title="Library Book Management API",
    description="RESTful API to manage library books with CRUD and partial update
    version="1.0.0"
)

# ----- Data Models -----
class Book(BaseModel):
    id: int
    title: str
    author: str
    year: int
    available: bool = True

class BookUpdate(BaseModel):
    title: Optional[str] = None
    author: Optional[str] = None
    year: Optional[int] = None
    available: Optional[bool] = None

# In-memory "database"
books_db: Dict[int, Book] = {}

# ----- Endpoints -----

@app.get("/books")
def get_books():
    """
    Retrieve all books in the library.
    """
    return list(books_db.values())
```

**OUTPUT:**

```
PS D:\BTECH\AI Assisted Coding\LABS ASSIGNMENTS\Lab 15> uvicorn lab15_libr
ary_api:app --reload
>>
INFO:     Will watch for changes in these directories: ['D:\\BTECH\\AI Ass
isted Coding\\LABS ASSIGNMENTS\\Lab 15']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [10024] using StatReload
INFO:     Started server process [17752]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:53788 - "GET /docs HTTP/1.1" 200 OK
INFO:     127.0.0.1:53788 - "GET /openapi.json HTTP/1.1" 200 OK
INFO:     127.0.0.1:49944 - "POST /books HTTP/1.1" 200 OK
```

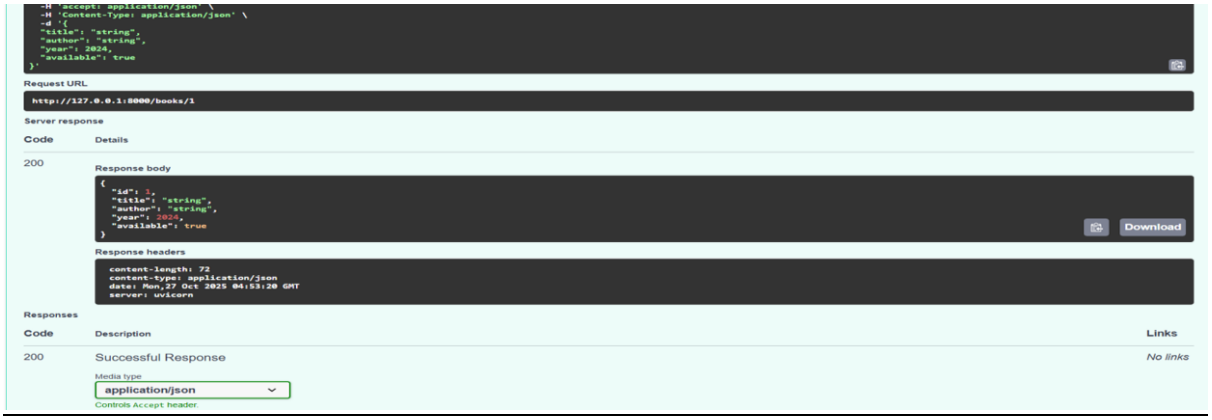
## Post:


-d '{
"id": 1,
"title": "Java",
"author": "Ramu",
"year": 2025,
"available": true
}'
Request URL
http://127.0.0.1:8000/books
Server response
Code
Details
200
Response body
{
"id": 1,
"title": "Java",
"author": "Ramu",
"year": 2025,
"available": true
}
Download
Response headers
content-length: 68
content-type: application/json
date: Mon,27 Oct 2025 04:46:19 GMT
server: uvicorn
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.

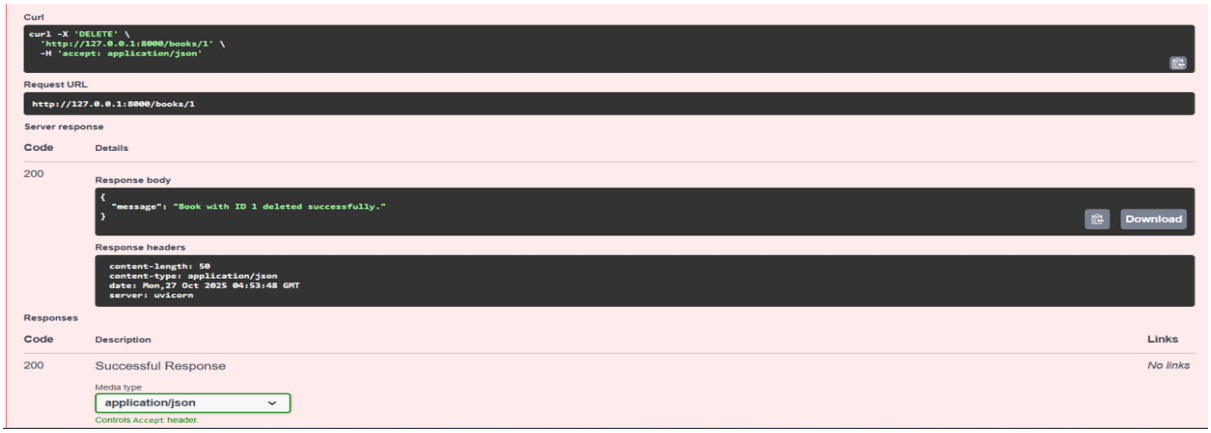
## Get:


Request URL
http://127.0.0.1:8000/books
Server response
Code
Details
200
Response body
[
{
"id": 1,
"title": "Java",
"author": "Ramu",
"year": 2025,
"available": true
}
]
Download
Response headers
content-length: 70
content-type: application/json
date: Mon,27 Oct 2025 04:51:58 GMT
server: uvicorn
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
"string"

## Put:


-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
"title": "string",
"author": "string",
"year": 2024,
"available": true
}'
Request URL
http://127.0.0.1:8000/books/1
Server response
Code
Details
200
Response body
{
"id": 1,
"title": "string",
"author": "string",
"year": 2024,
"available": true
}
Download
Response headers
content-length: 72
content-type: application/json
date: Mon,27 Oct 2025 04:53:20 GMT
server: uvicorn
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.

**Delete:**




**OBSERVATION:**

The library book management api efficiently handles basic operations like adding, viewing, updating, and deleting book records. it also supports partial updates using the patch method, making it flexible for managing book availability. error handling ensures smooth functioning even with invalid inputs. overall, it provides a simple and effective way to manage library data using restful principles.

# TASK-3

**QUESTION:**

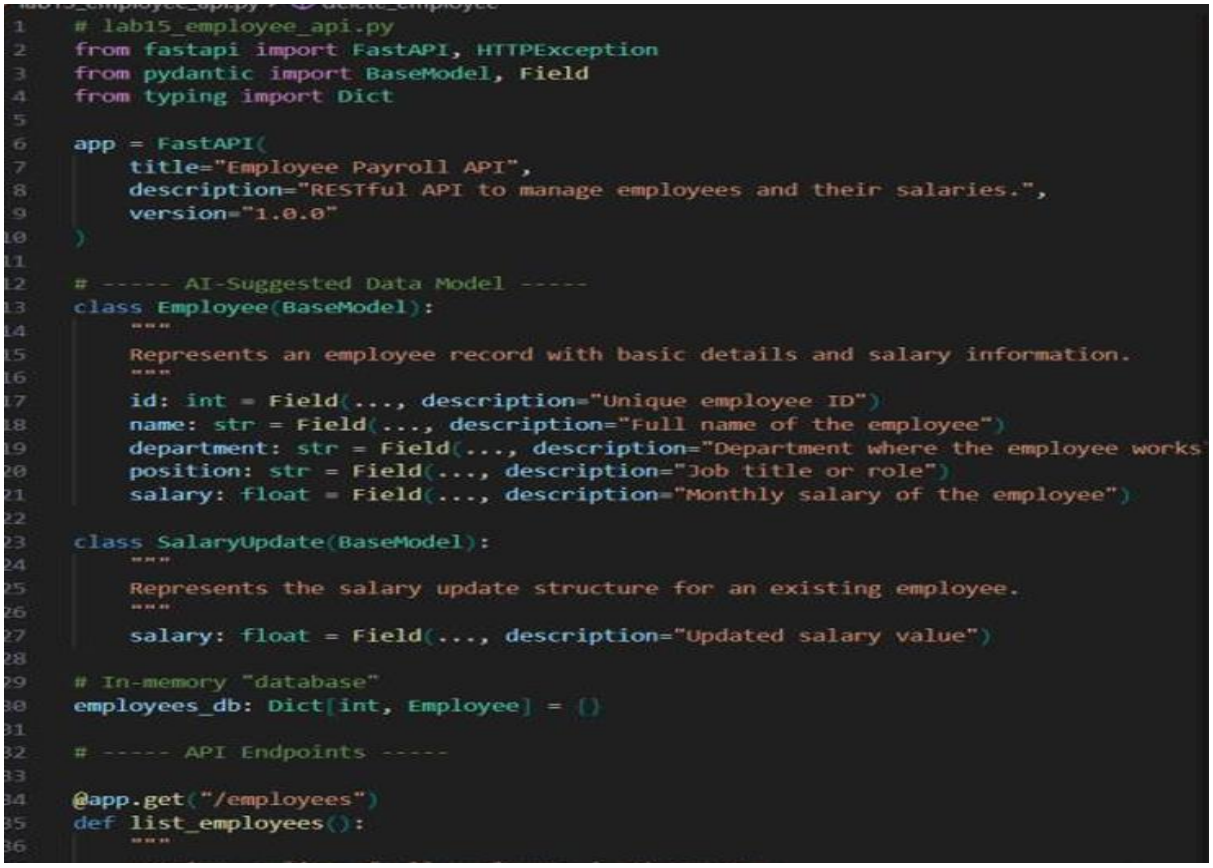
3 – Employee Payroll API
Task:
Create an API for managing employees and their salaries.
Instructions:
• Endpoints required:
o GET /employees → List all employees
o POST /employees → Add a new employee with salary

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.
Expected Output:
• API supporting salary management with clear documentation.

**PROMPT:**

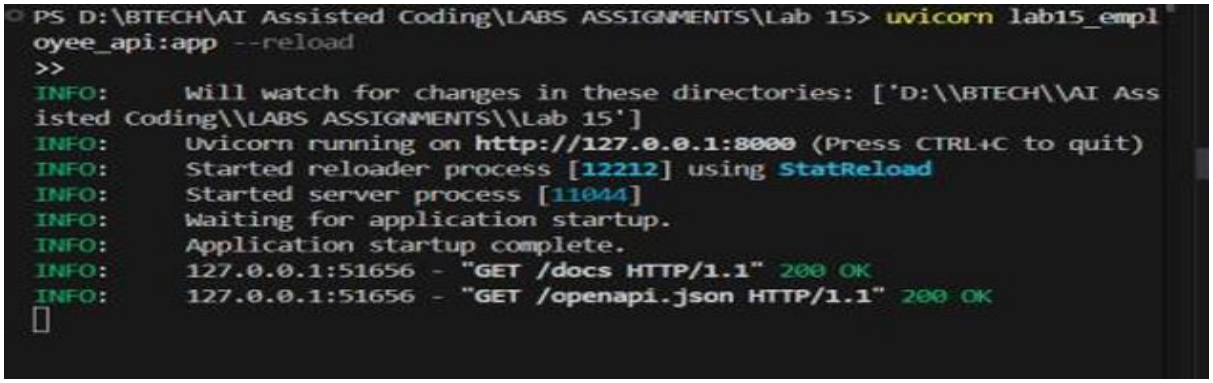
Create a python fastapi restful api for employee payroll management using an in memory list. include endpoints for get, post, put and delete employees. use ai to suggest a data model structure and add clear comments or docstrings for each endpoint. return responses in json format.

**CODE:**



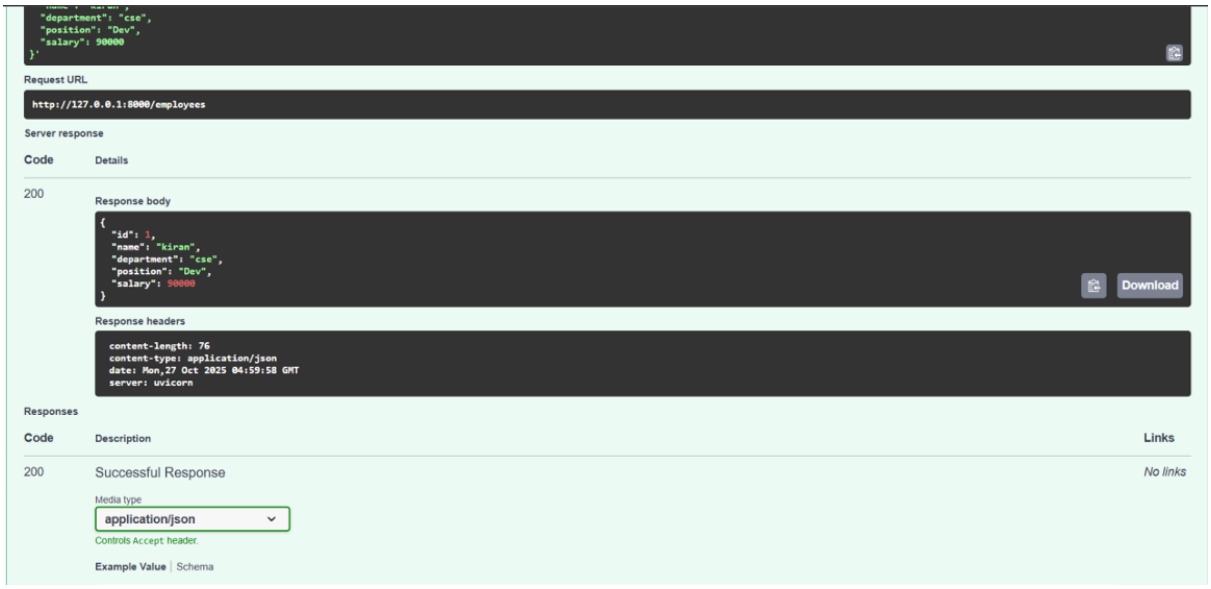
```python
# lab15_employee_api.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
from typing import Dict

app = FastAPI(
    title="Employee Payroll API",
    description="RESTful API to manage employees and their salaries.",
    version="1.0.0"
)

# ----- AI-Suggested Data Model -----
class Employee(BaseModel):
    """
    Represents an employee record with basic details and salary information.
    """
    id: int = Field(..., description="Unique employee ID")
    name: str = Field(..., description="Full name of the employee")
    department: str = Field(..., description="Department where the employee works")
    position: str = Field(..., description="Job title or role")
    salary: float = Field(..., description="Monthly salary of the employee")

class SalaryUpdate(BaseModel):
    """
    Represents the salary update structure for an existing employee.
    """
    salary: float = Field(..., description="Updated salary value")

# In-memory "database"
employees_db: Dict[int, Employee] = {}

# ----- API Endpoints -----

@app.get("/employees")
def list_employees():
    """
```

**OUTPUT:**

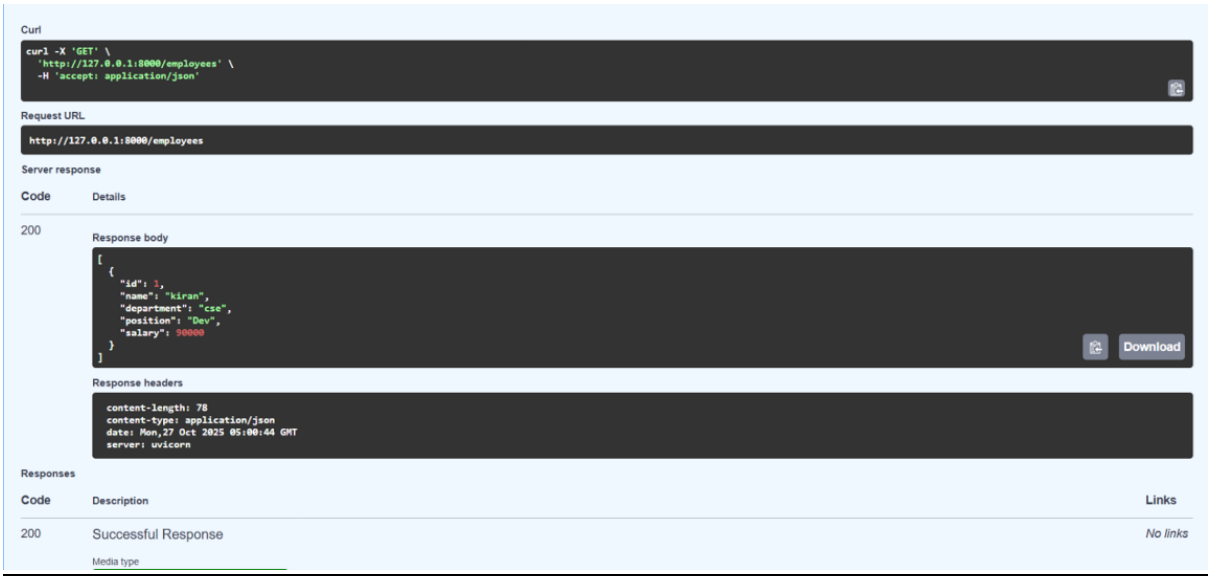


```
PS D:\BTECH\AI Assisted Coding\LABS ASSIGNMENTS\Lab 15> uvicorn lab15_empl
oyee_api:app --reload
>>
INFO:     Will watch for changes in these directories: ['D:\\BTECH\\AI Ass
isted Coding\\LABS ASSIGNMENTS\\Lab 15']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [12212] using StatReload
INFO:     Started server process [11044]
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
```

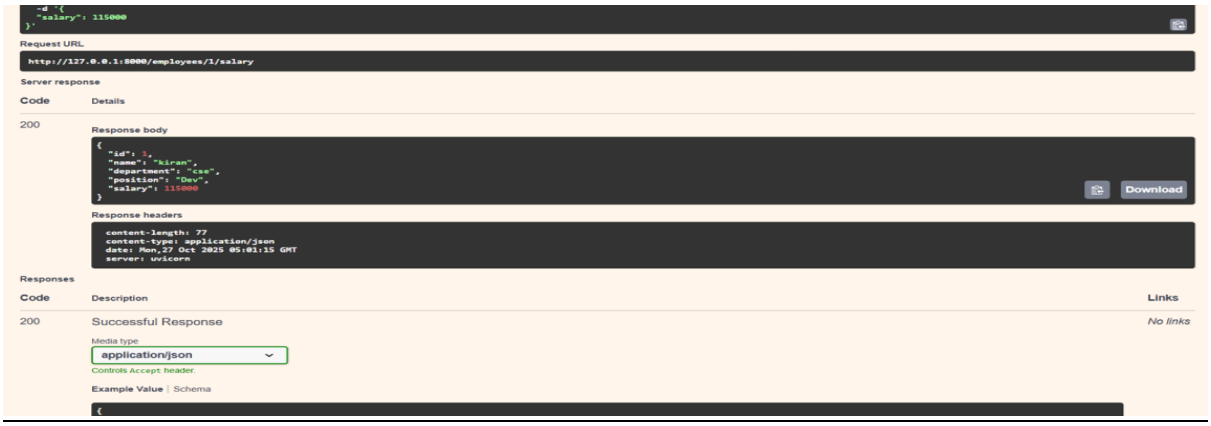
## Post:


"department": "cse",
"position": "Dev",
"salary": 90000
}'
Request URL
http://127.0.0.1:8000/employees
Server response
Code
Details
200
Response body
{
"id": 1,
"name": "kiran",
"department": "cse",
"position": "Dev",
"salary": 90000
}
Download
Response headers
content-length: 76
content-type: application/json
date: Mon,27 Oct 2025 04:59:58 GMT
server: uvicorn
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.
Example Value | Schema

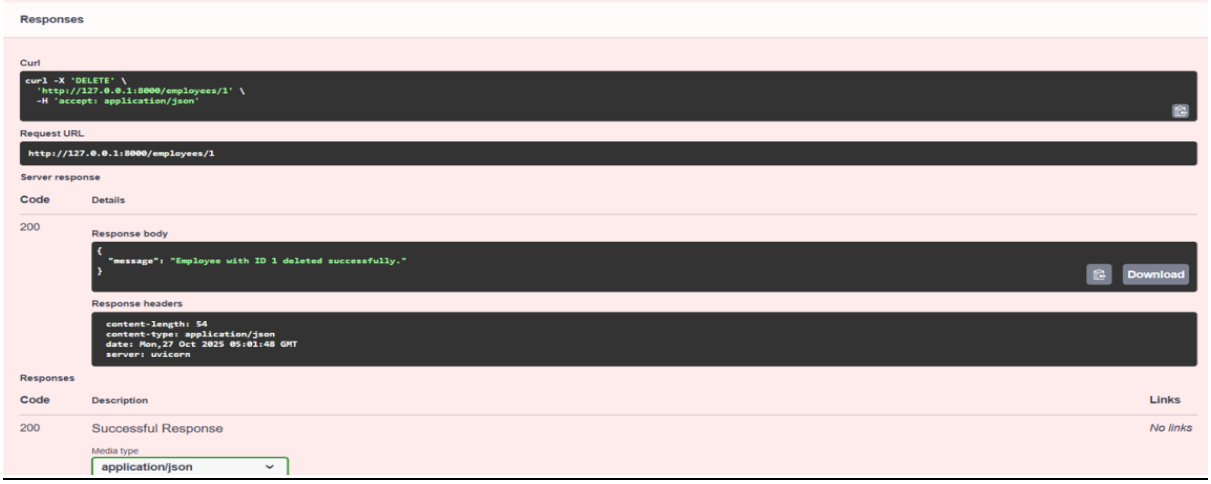
## Get:


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
Response body
[
{
"id": 1,
"name": "kiran",
"department": "cse",
"position": "Dev",
"salary": 90000
}
]
Download
Response headers
content-length: 78
content-type: application/json
date: Mon,27 Oct 2025 05:00:44 GMT
server: uvicorn
Responses
Code
Description
Links
200
Successful Response
No links
Media type

## Put:


-d '{
"salary": 115000
}'
Request URL
http://127.0.0.1:8000/employees/1/salary
Server response
Code
Details
200
Response body
{
"id": 1,
"name": "kiran",
"department": "cse",
"position": "Dev",
"salary": 115000
}
Download
Response headers
content-length: 77
content-type: application/json
date: Mon,27 Oct 2025 05:01:15 GMT
server: uvicorn
Responses
Code
Description
Links
200
Successful Response
No links
Media type
application/json
Controls Accept header.
Example Value | Schema
{

**Delete:**




**OBSERVATION:**

The employee payroll api effectively manages employee details and salary information. it allows adding, viewing, updating, and deleting employee records with ease. ai-generated data model suggestions and docstrings make the code well-documented and beginner friendly. overall, it provides a clear and organized approach to handling payroll data through restful api design.

# TASK-4

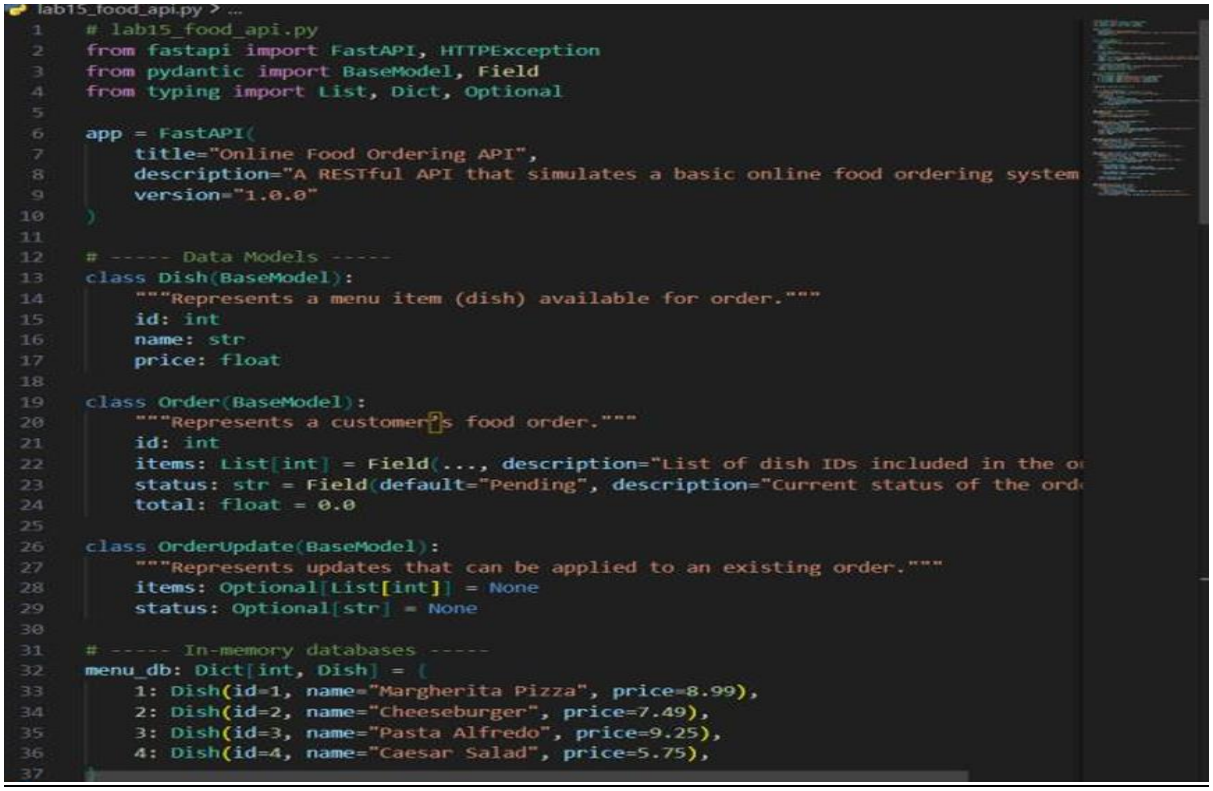
**QUESTION:**

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)
Expected Output:
• Fully working API simulating a food ordering backend

**PROMPT:**

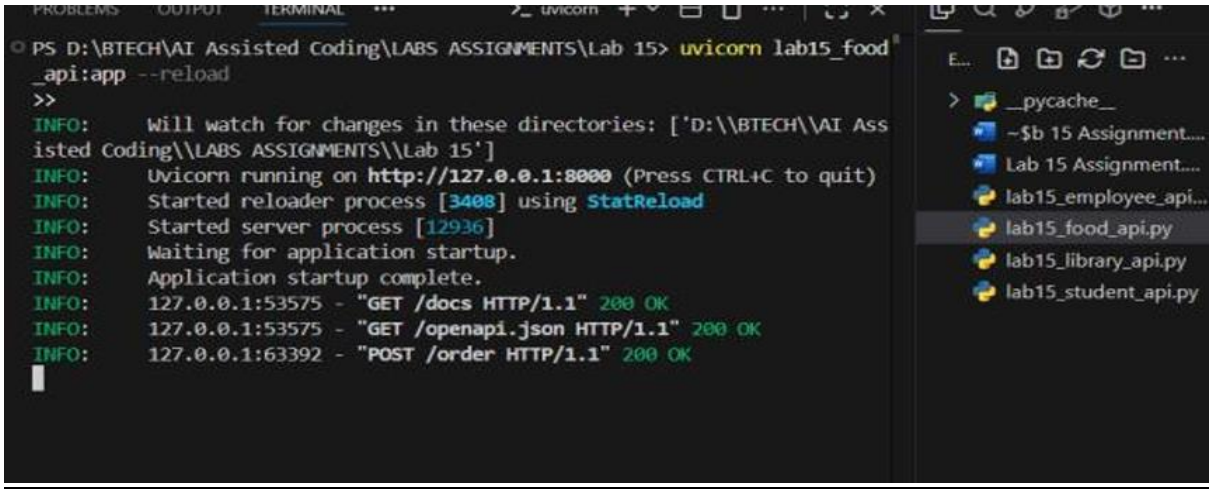
Create a python fastapi restful api for an online food ordering system using an in memory list. include endpoints for get menu, post order, get order by id, put to update order, and delete to cancel order. let ai suggest improvements like authentication and pagination. return all responses in json format.

**CODE:**



```python
# lab15_food_api.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
from typing import List, Dict, Optional

app = FastAPI(
    title="Online Food Ordering API",
    description="A RESTful API that simulates a basic online food ordering system",
    version="1.0.0"
)

# ----- Data Models -----
class Dish(BaseModel):
    """Represents a menu item (dish) available for order."""
    id: int
    name: str
    price: float

class Order(BaseModel):
    """Represents a customer's food order."""
    id: int
    items: List[int] = Field(..., description="List of dish IDs included in the o
    status: str = Field(default="Pending", description="Current status of the ord
    total: float = 0.0

class OrderUpdate(BaseModel):
    """Represents updates that can be applied to an existing order."""
    items: Optional[List[int]] = None
    status: Optional[str] = None

# ----- In-memory databases -----
menu_db: Dict[int, Dish] = {
    1: Dish(id=1, name="Margherita Pizza", price=8.99),
    2: Dish(id=2, name="Cheeseburger", price=7.49),
    3: Dish(id=3, name="Pasta Alfredo", price=9.25),
    4: Dish(id=4, name="Caesar Salad", price=5.75),
```

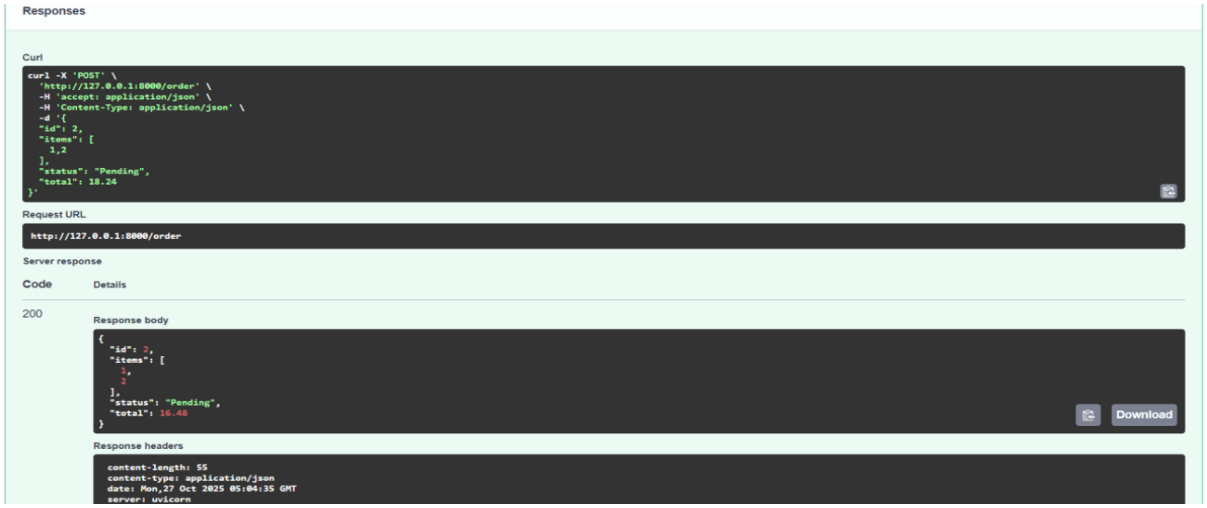
**OUTPUT:**



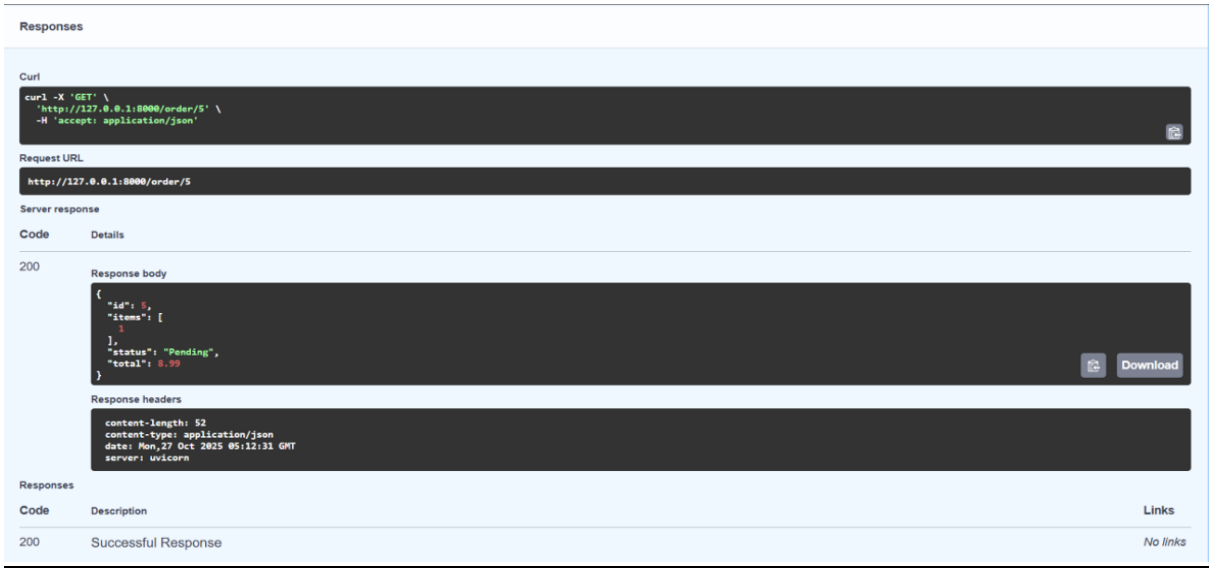
```
PS D:\BTECH\AI Assisted Coding\LABS ASSIGNMENTS\Lab 15> uvicorn lab15_food
_api:app --reload
>>
INFO:     Will watch for changes in these directories: ['D:\\BTECH\\AI Ass
isted Coding\\LABS ASSIGNMENTS\\Lab 15']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [3408] using StatReload
INFO:     Started server process [12936]
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
```

```
__pycache__
~$b 15 Assignment....
Lab 15 Assignment....
lab15_employee_api...
lab15_food_api.py
lab15_library_api.py
lab15_student_api.py
```

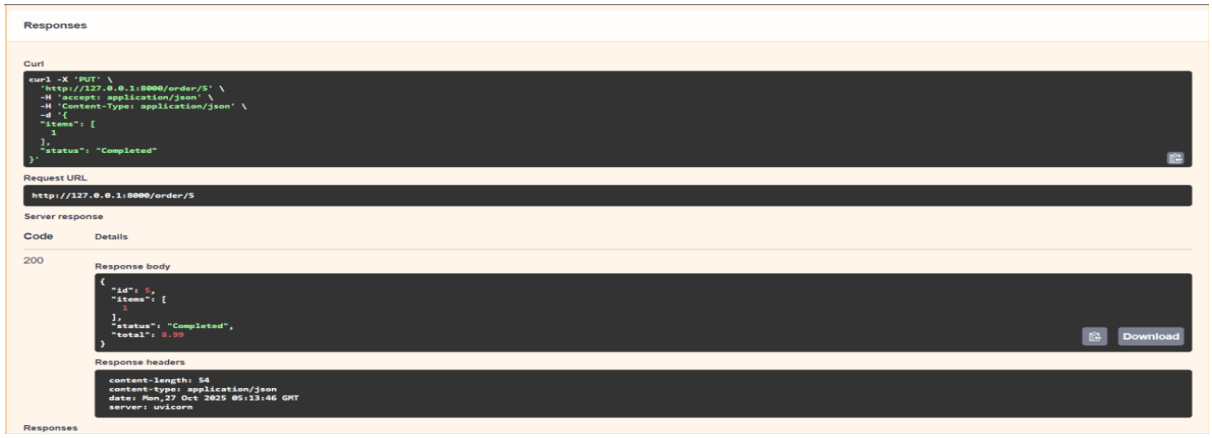
## Post:


Responses
Curl
curl -X 'POST' \
'http://127.0.0.1:8000/order' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
"id": 2,
"items": [
1,2
],
"status": "Pending",
"total": 18.24
}'
Request URL
http://127.0.0.1:8000/order
Server response
Code
Details
200
Response body
{
"id": 2,
"items": [
1,
2
],
"status": "Pending",
"total": 16.48
}
Download
Response headers
content-length: 55
content-type: application/json
date: Mon,27 Oct 2025 05:04:35 GMT
server: uvicorn

## Get:


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
Response body
{
"id": 5,
"items": [
1
],
"status": "Pending",
"total": 8.99
}
Download
Response headers
content-length: 52
content-type: application/json
date: Mon,27 Oct 2025 05:12:31 GMT
server: uvicorn
Responses
Code
Description
Links
200
Successful Response
No links

## 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
Response body
{
"id": 5,
"items": [
1
],
"status": "Completed",
"total": 8.99
}
Download
Response headers
content-length: 54
content-type: application/json
date: Mon,27 Oct 2025 05:13:46 GMT
server: uvicorn
Responses

**Delete:**


**OBSERVATION:**

The online food ordering api successfully simulates a real-time backend system. it allows users to view the menu, place new orders, track order status, update orders, and cancel them easily. the api demonstrates clear structure and smooth functionality. ai-suggested features like authentication and pagination can further enhance security and performance. overall, it provides a practical example of how a modern food ordering system can be managed through restful api design.