

AI ASSISTED CODING

NAME : BAYANA ARJUN

ROLL NO: 2403A510A9

ASSIGNMENT : 15.1

#TASK-1

PROMPT: 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

CODE:

```

app.py > ...
1  from flask import Flask, jsonify, request
2
3  app = Flask(__name__)
4
5  # In-memory storage (list of dicts)
6  students = [
7      {"id": 1, "name": "Priya", "age": 21, "course": "CSE"},
8      {"id": 2, "name": "Rahul", "age": 22, "course": "ECE"}
9  ]
10
11 # GET - List all students
12 @app.route('/students', methods=['GET'])
13 def get_students():
14     return jsonify(students), 200
15
16 # POST - Add a new student
17 @app.route('/students', methods=['POST'])
18 def add_student():
19     new_student = request.get_json()
20     new_student["id"] = students[-1]["id"] + 1 if students else 1
21     students.append(new_student)
22     return jsonify({"message": "Student added successfully!", "student": new_student}), 201
23
24 # PUT - Update student details by ID
25 @app.route('/students/<int:student_id>', methods=['PUT'])
26 def update_student(student_id):
27     student = next((s for s in students if s["id"] == student_id), None)
28     if not student:
29         return jsonify({"error": "Student not found"}), 404
30
31     updated_data = request.get_json()
32     student.update(updated_data)
33     return jsonify({"message": "Student updated successfully!", "student": student}), 200
34
35 # DELETE - Delete student by ID
36 @app.route('/students/<int:student_id>', methods=['DELETE'])
37 def delete_student(student_id):

```

```

app.py > ...
24 # PUT - Update student details by ID
25 @app.route('/students/<int:student_id>', methods=['PUT'])
26 def update_student(student_id):
27     student = next((s for s in students if s["id"] == student_id), None)
28     if not student:
29         return jsonify({"error": "Student not found"}), 404
30
31     updated_data = request.get_json()
32     student.update(updated_data)
33     return jsonify({"message": "Student updated successfully!", "student": student}), 200
34
35 # DELETE - Delete student by ID
36 @app.route('/students/<int:student_id>', methods=['DELETE'])
37 def delete_student(student_id):
38     global students
39     students = [s for s in students if s["id"] != student_id]
40     return jsonify({"message": "Student deleted successfully!"}), 200
41
42 if __name__ == '__main__':
43     app.run(debug=True)
44

```

OUTPUT :

```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS

/Users/keerthi priya/Desktop/studentApi/testcases.py
PS C:\Users\keerthi priya\Desktop\studentApi>
> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe"
/Users/keerthi priya/Desktop/studentApi/app.py"

```

```
PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS

* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 659-110-340
```

OBSERVATION:

Flask server starts successfully on <http://127.0.0.1:5000>.

Returns all student records as a JSON array.

Initial data is the hardcoded list of students.

HTTP status code: 200 OK.

All student records are stored in a Python list.

#TASK-2

PROMPT:

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:

```

app1.py > ...
1 from flask import Flask, jsonify, request, abort
2
3 app = Flask(__name__)
4
5 # In-memory database simulation
6 books = [
7     {"id": 1, "title": "1984", "author": "George Orwell", "available": True},
8     {"id": 2, "title": "To Kill a Mockingbird", "author": "Harper Lee", "available": True},
9 ]
10
11 # Helper function to find book by id
12 def find_book(book_id):
13     return next((book for book in books if book["id"] == book_id), None)
14
15 # GET /books → Retrieve all books
16 @app.route("/books", methods=["GET"])
17 def get_books():
18     return jsonify(books), 200
19
20 # POST /books → Add a new book
21 @app.route("/books", methods=["POST"])
22 def add_book():
23     if not request.json or "title" not in request.json or "author" not in request.json:
24         abort(400, description="Invalid request: title and author required")
25
26     new_id = max(book["id"] for book in books) + 1 if books else 1
27     book = {
28         "id": new_id,
29         "title": request.json["title"],
30         "author": request.json["author"],
31         "available": request.json.get("available", True)
32     }
33     books.append(book)
34     return jsonify(book), 201
35
36 # GET /books/{id} → Get details of a specific book
37 @app.route("/books/<int:book_id>", methods=["GET"])

```

Ln 80, Col 1, Spaces: 4, LTF: 8, CRLF, Python 3.11.0

```

app1.py > ...
37 @app.route("/books/<int:book_id>", methods=["GET"])
38 def get_book(book_id):
39     book = find_book(book_id)
40     if not book:
41         abort(404, description="Book not found")
42     return jsonify(book), 200
43
44 # PATCH /books/{id} → Update partial book details
45 @app.route("/books/<int:book_id>", methods=["PATCH"])
46 def update_book(book_id):
47     book = find_book(book_id)
48     if not book:
49         abort(404, description="Book not found")
50
51     data = request.json
52     if not data:
53         abort(400, description="Invalid request: no data provided")
54
55     # Only update fields provided in request
56     book.update({k: v for k, v in data.items() if k in ["title", "author", "available"]})
57     return jsonify(book), 200
58
59 # DELETE /books/{id} → Remove a book
60 @app.route("/books/<int:book_id>", methods=["DELETE"])
61 def delete_book(book_id):
62     book = find_book(book_id)
63     if not book:
64         abort(404, description="Book not found")
65     books.remove(book)
66     return jsonify({"message": "Book deleted"}), 200
67
68 # Error handling
69 @app.errorhandler(400)
70 def bad_request(error):
71     return jsonify({"error": "Bad Request", "message": error.description}), 400
72
73 @app.errorhandler(404)

```

```

72
73 @app.errorhandler(404)
74 def not_found(error):
75     return jsonify({"error": "Not Found", "message": error.description}), 404
76
77 # Run the app
78 if __name__ == "__main__":
79     app.run(debug=True)

```

OUTPUT:

```

Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Restarting with stat
* Debugger is active!
* Debugger is active!
* Debugger PIN: 659-110-340
127.0.0.1 - - [25/Oct/2025 18:58:00] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [25/Oct/2025 18:58:00] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [25/Oct/2025 18:58:32] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [25/Oct/2025 18:58:32] "GET /favicon.ico HTTP/1.1" 404 -

```

OBSERVATION:

Adds a new book to the in-memory list.

Automatically generates id.

Optional field available defaults to True if not provided.

Response includes the new book object

#TASK-3

PROMPT :

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:

```

employee.py > ...
1  from fastapi import FastAPI, HTTPException
2  from pydantic import BaseModel, Field
3  from typing import List, Optional
4
5  app = FastAPI(title="Employee Salary Management API",
6               description="API to manage employees and their salaries",
7               version="1.0.0")
8
9  # -----
10 # AI-suggested Data Model
11 # -----
12 class Employee(BaseModel):
13     """
14     Employee data model.
15     id: Unique identifier for each employee
16     name: Employee full name
17     position: Job title of the employee
18     salary: Employee's current salary
19     """
20     id: int
21     name: str = Field(..., example="John Doe")
22     position: str = Field(..., example="Software Engineer")
23     salary: float = Field(..., example=50000.0)
24
25
26 # In-memory "database"
27 employees: List[Employee] = []
28
29 # -----
30 # Endpoints
31 # -----
32
33 @app.get("/employees", response_model=List[Employee])

```



```

employee.py > ...
34 def list_employees():
35     """
36     Retrieve the list of all employees.
37     Returns a list of employee records with salary details.
38     """
39     return employees
40
41
42 @app.post("/employees", response_model=Employee, status_code=201)
43 def add_employee(employee: Employee):
44     """
45     Add a new employee to the system.
46     Expects an employee object with id, name, position, and salary.
47     Raises 400 if an employee with same id already exists.
48     """
49     if any(emp.id == employee.id for emp in employees):
50         raise HTTPException(status_code=400, detail="Employee with this ID already exists")
51     employees.append(employee)
52     return employee
53
54
55 @app.put("/employees/{employee_id}/salary", response_model=Employee)
56 def update_salary(employee_id: int, new_salary: float):
57     """
58     Update the salary of an existing employee.
59     Parameters:
60     - employee_id: ID of the employee
61     - new_salary: New salary amount
62     Raises 404 if employee is not found.
63     """
64     for emp in employees:
65         if emp.id == employee_id:
66             emp.salary = new_salary
67
68 def update_salary(employee_id: int, new_salary: float):
69     if emp.id == employee_id:
70         emp.salary = new_salary
71         return emp
72     raise HTTPException(status_code=404, detail="Employee not found")
73
74
75 @app.delete("/employees/{employee_id}", response_model=dict)
76 def delete_employee(employee_id: int):
77     """
78     Remove an employee from the system.
79     Parameters:
80     - employee_id: ID of the employee to remove
81     Returns a confirmation message.
82     Raises 404 if employee is not found.
83     """
84     for emp in employees:
85         if emp.id == employee_id:
86             employees.remove(emp)
87             return {"message": f"Employee {employee_id} deleted successfully"}
88     raise HTTPException(status_code=404, detail="Employee not found")
89

```

OUTPUT:


```

task4.py > ...
1  from fastapi import FastAPI, HTTPException
2  from pydantic import BaseModel, Field
3  from typing import List, Optional
4
5  app = FastAPI(title="Online Food Ordering API", version="1.0")
6
7  # -----
8  # Data Models
9  # -----
10 class Dish(BaseModel):
11     id: int
12     name: str
13     price: float
14     available: bool = True
15
16 class OrderItem(BaseModel):
17     dish_id: int
18     quantity: int
19
20 class Order(BaseModel):
21     id: int
22     items: List[OrderItem]
23     status: str = "Pending" # Possible statuses: Pending, Preparing, Delivered
24
25 # -----
26 # In-memory "database"
27 # -----
28 menu = [
29     Dish(id=1, name="Burger", price=5.99),
30     Dish(id=2, name="Pizza", price=8.99),
31     Dish(id=3, name="Pasta", price=7.99),
32 ]
33
34 orders: List[Order] = []
35
36 # -----
37 # Endpoints

```

```

task4.py > ...
37 # Endpoints
38 # -----
39
40 @app.get("/menu", response_model=List[Dish])
41 def get_menu():
42     """Retrieve the list of available dishes."""
43     return menu
44
45 @app.post("/order", response_model=Order, status_code=201)
46 def place_order(order: Order):
47     """Place a new order with items."""
48     if any(existing.id == order.id for existing in orders):
49         raise HTTPException(status_code=400, detail="Order ID already exists")
50     orders.append(order)
51     return order
52
53 @app.get("/order/{order_id}", response_model=Order)
54 def track_order(order_id: int):
55     """Get order status for a specific order ID."""
56     for order in orders:
57         if order.id == order_id:
58             return order
59     raise HTTPException(status_code=404, detail="Order not found")
60
61 @app.put("/order/{order_id}", response_model=Order)
62 def update_order(order_id: int, updated_order: Order):
63     """Update an existing order (change items)."""
64     for idx, order in enumerate(orders):
65         if order.id == order_id:
66             orders[idx] = updated_order
67             return updated_order
68     raise HTTPException(status_code=404, detail="Order not found")
69
70 @app.delete("/order/{order_id}", response_model=dict)
71 def cancel_order(order_id: int):
72     """Cancel an existing order."""
73     for order in orders:

```

```

2 def update_order(order_id: int, updated_order: Order):
3     for idx, order in enumerate(orders):
4         if order.id == order_id:
5             orders[idx] = updated_order
6             return updated_order
7     raise HTTPException(status_code=404, detail="Order not found")
8
9
10 @app.delete("/order/{order_id}", response_model=dict)
11 def cancel_order(order_id: int):
12     """Cancel an existing order."""
13     for order in orders:
14         if order.id == order_id:
15             orders.remove(order)
16             return {"message": f"Order {order_id} canceled successfully"}
17     raise HTTPException(status_code=404, detail="Order not found")
18

```

```

testcases.py > ...
1  def test_get_menu():
2      response = client.get("/menu")
3      assert response.status_code == 200
4      assert len(response.json()) >= 3
5
6  def test_place_order():
7      order = {"id": 1, "items": [{"dish_id": 1, "quantity": 2}]}
8      response = client.post("/order", json=order)
9      assert response.status_code == 201
10     assert response.json()["status"] == "Pending"
11
12  def test_track_order():
13     response = client.get("/order/1")
14     assert response.status_code == 200
15     assert response.json()["id"] == 1
16
17  def test_update_order():
18     updated_order = {"id": 1, "items": [{"dish_id": 2, "quantity": 1}], "status": "Pending"}
19     response = client.put("/order/1", json=updated_order)
20     assert response.status_code == 200
21     assert response.json()["items"][0]["dish_id"] == 2
22
23  def test_cancel_order():
24     response = client.delete("/order/1")
25     assert response.status_code == 200
26     assert response.json()["message"] == "Order 1 canceled successfully"
27

```

OUTPUT:

```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\keerthi priya\Desktop\studentApi> & "C:/Users/keerthi priya/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/keerthi p
riya/Desktop/studentApi/task4.py"
PS C:\Users\keerthi priya\Desktop\studentApi>

```

OBSERVATION:

Menu Retrieval Works – GET /menu correctly lists all available dishes.

Order Placement Works – POST /order allows creating a new order with items.

Order Tracking Works – GET /order/{id} returns the correct order and status.

Order Update Works – PUT /order/{id} successfully updates order items.

Order Cancellation Works – DELETE /order/{id} removes the order and confirms deletion.