AI ASSISTED CODING

NAME : DUGYALA ASHMITHA

ROLL NO: 2403A510G5

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 > ...

```python
from flask import Flask, jsonify, request

app = Flask(__name__)

# In-memory storage (list of dicts)
students = [
    {"id": 1, "name": "Priya", "age": 21, "course": "CSE"},
    {"id": 2, "name": "Rahul", "age": 22, "course": "ECE"}
]

# GET - List all students
@app.route('/students', methods=['GET'])
def get_students():
    return jsonify(students), 200

# POST - Add a new student
@app.route('/students', methods=['POST'])
def add_student():
    new_student = request.get_json()
    new_student["id"] = students[-1]["id"] + 1 if students else 1
    students.append(new_student)
    return jsonify({"message": "Student added successfully!", "student": new_student}), 201

# PUT - Update student details by ID
@app.route('/students/<int:student_id>', methods=['PUT'])
def update_student(student_id):
    student = next((s for s in students if s["id"] == student_id), None)
    if not student:
        return jsonify({"error": "Student not found"}), 404

    updated_data = request.get_json()
    student.update(updated_data)
    return jsonify({"message": "Student updated successfully!", "student": student}), 200

# DELETE - Delete student by ID
@app.route('/students/<int:student_id>', methods=['DELETE'])
def delete_student(student_id):
```

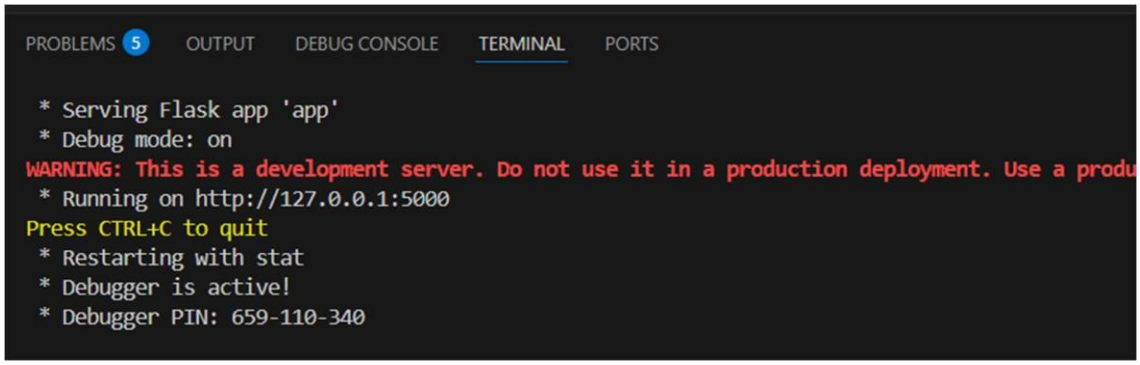
Ln 44, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.11.9 (Mi

app.py > ...

```python
# PUT - Update student details by ID
@app.route('/students/<int:student_id>', methods=['PUT'])
def update_student(student_id):
    student = next((s for s in students if s["id"] == student_id), None)
    if not student:
        return jsonify({"error": "Student not found"}), 404

    updated_data = request.get_json()
    student.update(updated_data)
    return jsonify({"message": "Student updated successfully!", "student": student}), 200

# DELETE - Delete student by ID
@app.route('/students/<int:student_id>', methods=['DELETE'])
def delete_student(student_id):
    global students
    students = [s for s in students if s["id"] != student_id]
    return jsonify({"message": "Student deleted successfully!"}), 200

if __name__ == '__main__':
    app.run(debug=True)
```

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"
```



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 > ...

```python
from flask import Flask, jsonify, request, abort

app = Flask(__name__)

# In-memory database simulation
books = [
    {"id": 1, "title": "1984", "author": "George Orwell", "available": True},
    {"id": 2, "title": "To Kill a Mockingbird", "author": "Harper Lee", "available": True},
]

# Helper function to find book by id
def find_book(book_id):
    return next((book for book in books if book["id"] == book_id), None)

# GET /books → Retrieve all books
@app.route("/books", methods=["GET"])
def get_books():
    return jsonify(books), 200

# POST /books → Add a new book
@app.route("/books", methods=["POST"])
def add_book():
    if not request.json or "title" not in request.json or "author" not in request.json:
        abort(400, description="Invalid request: title and author required")

    new_id = max(book["id"] for book in books) + 1 if books else 1
    book = {
        "id": new_id,
        "title": request.json["title"],
        "author": request.json["author"],
        "available": request.json.get("available", True)
    }
    books.append(book)
    return jsonify(book), 201

# GET /books/{id} → Get details of a specific book
@app.route("/books/<int:book_id>", methods=["GET"])
```

Ln 80, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.11.0

app1.py > ...

```python
def get_book(book_id):
    book = find_book(book_id)
    if not book:
        abort(404, description="Book not found")
    return jsonify(book), 200

# PATCH /books/{id} → Update partial book details
@app.route("/books/<int:book_id>", methods=["PATCH"])
def update_book(book_id):
    book = find_book(book_id)
    if not book:
        abort(404, description="Book not found")

    data = request.json
    if not data:
        abort(400, description="Invalid request: no data provided")

    # Only update fields provided in request
    book.update({k: v for k, v in data.items() if k in ["title", "author", "available"]})
    return jsonify(book), 200

# DELETE /books/{id} → Remove a book
@app.route("/books/<int:book_id>", methods=["DELETE"])
def delete_book(book_id):
    book = find_book(book_id)
    if not book:
        abort(404, description="Book not found")
    books.remove(book)
    return jsonify({"message": "Book deleted"}), 200

# Error handling
@app.errorhandler(400)
def bad_request(error):
    return jsonify({"error": "Bad Request", "message": error.description}), 400

@app.errorhandler(404)
```

```python

@app.errorhandler(404)
def not_found(error):
    return jsonify({"error": "Not Found", "message": error.description}), 404

# Run the app
if __name__ == "__main__":
    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:

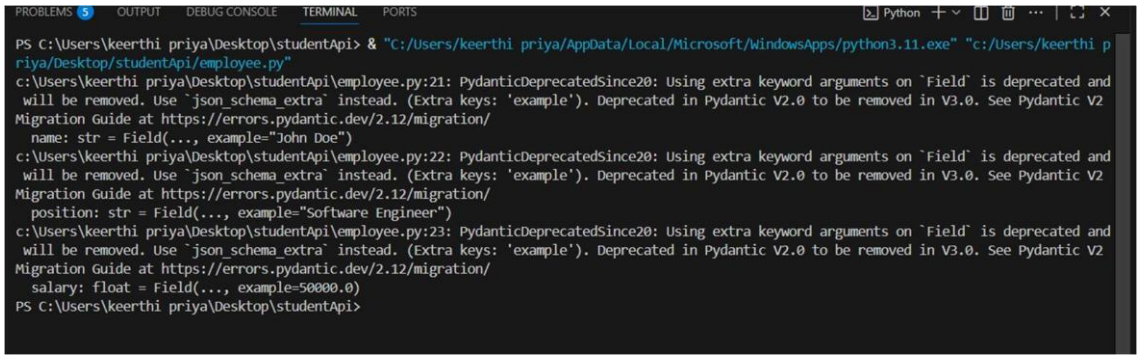
employee.py > ...

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
from typing import List, Optional

app = FastAPI(title="Employee Salary Management API",
              description="API to manage employees and their salaries",
              version="1.0.0")

# ---------------------------
# AI-suggested Data Model
# ---------------------------
class Employee(BaseModel):
    """
    Employee data model.
    id: Unique identifier for each employee
    name: Employee full name
    position: Job title of the employee
    salary: Employee's current salary
    """
    id: int
    name: str = Field(..., example="John Doe")
    position: str = Field(..., example="Software Engineer")
    salary: float = Field(..., example=50000.0)


# In-memory "database"
employees: List[Employee] = []

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

@app.get("/employees", response_model=List[Employee])
```

```python
def list_employees():
    """
    Retrieve the list of all employees.
    Returns a list of employee records with salary details.
    """
    return employees


@app.post("/employees", response_model=Employee, status_code=201)
def add_employee(employee: Employee):
    """
    Add a new employee to the system.
    Expects an employee object with id, name, position, and salary.
    Raises 400 if an employee with same id already exists.
    """
    if any(emp.id == employee.id for emp in employees):
        raise HTTPException(status_code=400, detail="Employee with this ID already exists")
    employees.append(employee)
    return employee


@app.put("/employees/{employee_id}/salary", response_model=Employee)
def update_salary(employee_id: int, new_salary: float):
    """
    Update the salary of an existing employee.
    Parameters:
    - employee_id: ID of the employee
    - new_salary: New salary amount
    Raises 404 if employee is not found.
    """
    for emp in employees:
        if emp.id == employee_id:
            emp.salary = new_salary
```

```python
def update_salary(employee_id: int, new_salary: float):
        if emp.id == employee_id:
            emp.salary = new_salary
            return emp
    raise HTTPException(status_code=404, detail="Employee not found")


@app.delete("/employees/{employee_id}", response_model=dict)
def delete_employee(employee_id: int):
    """
    Remove an employee from the system.
    Parameters:
    - employee_id: ID of the employee to remove
    Returns a confirmation message.
    Raises 404 if employee is not found.
    """
    for emp in employees:
        if emp.id == employee_id:
            employees.remove(emp)
            return {"message": f"Employee {employee_id} deleted successfully"}
    raise HTTPException(status_code=404, detail="Employee not found")
```

OUTPUT:



OBSERVATION:

Employees are stored in a Python list (employees: List[Employee]).

Data is volatile — restarting the server resets the list.

Supports multiple employees with unique IDs.

#TASK-4

PROMPT:

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:

task4.py > ...

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
from typing import List, Optional

app = FastAPI(title="Online Food Ordering API", version="1.0")

# ---------------------------
# Data Models
# ---------------------------
class Dish(BaseModel):
    id: int
    name: str
    price: float
    available: bool = True

class OrderItem(BaseModel):
    dish_id: int
    quantity: int

class Order(BaseModel):
    id: int
    items: List[OrderItem]
    status: str = "Pending"  # Possible statuses: Pending, Preparing, Delivered

# ---------------------------
# In-memory "database"
# ---------------------------
menu = [
    Dish(id=1, name="Burger", price=5.99),
    Dish(id=2, name="Pizza", price=8.99),
    Dish(id=3, name="Pasta", price=7.99),
]

orders: List[Order] = []

# ---------------------------
# Endpoints
```

task4.py > ...

```python
# Endpoints
# ---------------------------

@app.get("/menu", response_model=List[Dish])
def get_menu():
    """Retrieve the list of available dishes."""
    return menu

@app.post("/order", response_model=Order, status_code=201)
def place_order(order: Order):
    """Place a new order with items."""
    if any(existing.id == order.id for existing in orders):
        raise HTTPException(status_code=400, detail="Order ID already exists")
    orders.append(order)
    return order

@app.get("/order/{order_id}", response_model=Order)
def track_order(order_id: int):
    """Get order status for a specific order ID."""
    for order in orders:
        if order.id == order_id:
            return order
    raise HTTPException(status_code=404, detail="Order not found")

@app.put("/order/{order_id}", response_model=Order)
def update_order(order_id: int, updated_order: Order):
    """Update an existing order (change items)."""
    for idx, order in enumerate(orders):
        if order.id == order_id:
            orders[idx] = updated_order
            return updated_order
    raise HTTPException(status_code=404, detail="Order not found")

@app.delete("/order/{order_id}", response_model=dict)
def cancel_order(order_id: int):
    """Cancel an existing order."""
    for order in orders:
```

```python
def update_order(order_id: int, updated_order: Order):
            orders[idx] = updated_order
            return updated_order
    raise HTTPException(status_code=404, detail="Order not found")

@app.delete("/order/{order_id}", response_model=dict)
def cancel_order(order_id: int):
    """Cancel an existing order."""
    for order in orders:
        if order.id == order_id:
            orders.remove(order)
            return {"message": f"Order {order_id} canceled successfully"}
    raise HTTPException(status_code=404, detail="Order not found")
```

testcases.py > ...

```python
def test_get_menu():
    response = client.get("/menu")
    assert response.status_code == 200
    assert len(response.json()) >= 3

def test_place_order():
    order = {"id": 1, "items": [{"dish_id": 1, "quantity": 2}]}
    response = client.post("/order", json=order)
    assert response.status_code == 201
    assert response.json()["status"] == "Pending"

def test_track_order():
    response = client.get("/order/1")
    assert response.status_code == 200
    assert response.json()["id"] == 1

def test_update_order():
    updated_order = {"id": 1, "items": [{"dish_id": 2, "quantity": 1}], "status": "Pending"}
    response = client.put("/order/1", json=updated_order)
    assert response.status_code == 200
    assert response.json()["items"][0]["dish_id"] == 2

def test_cancel_order():
    response = client.delete("/order/1")
    assert response.status_code == 200
    assert response.json()["message"] == "Order 1 canceled successfully"

```

OUTPUT:

```
PROBLEMS 5  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                   Python

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.