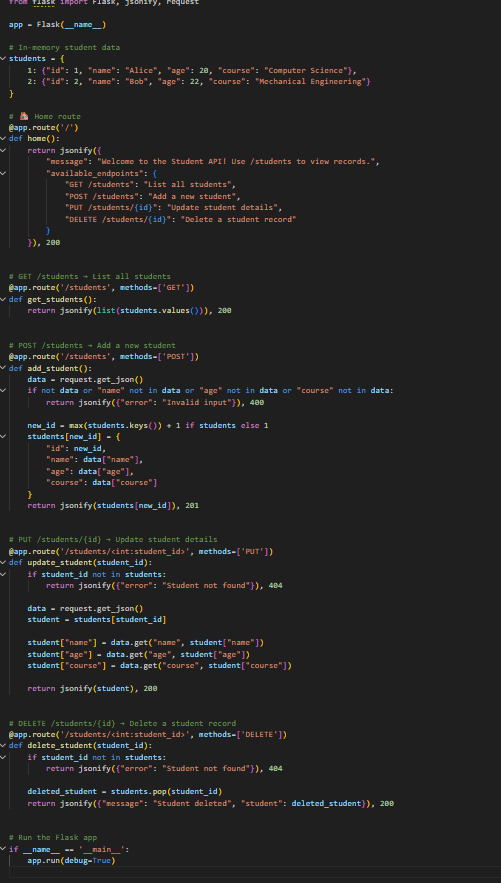
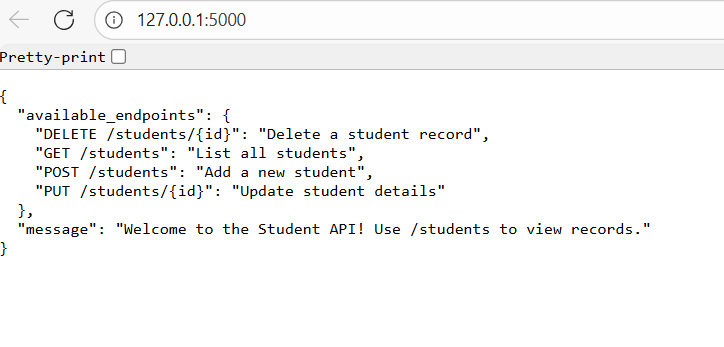
Task 1 – Student Records API:

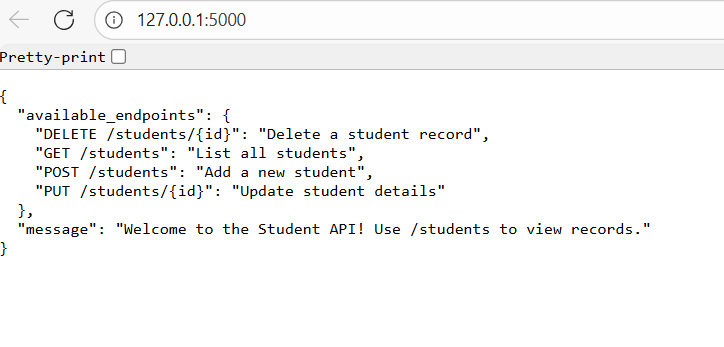
Prompt: Build a complete RESTful API using **Python and Flask** to manage student records.

**Requirements:**

* Use an **in-memory dictionary** to store student data.
* Include the following endpoints:
  + GET /students → List all students
  + POST /students → Add a new student
  + PUT /students/{id} → Update student details
  + DELETE /students/{id} → Delete a student
* Return **JSON responses** for all endpoints.
* Add a **home route (/)** that returns a welcome message and lists all available endpoints.
* Code:



Output: 

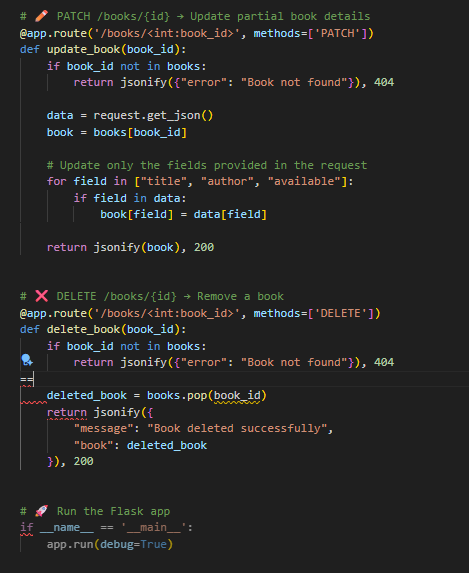


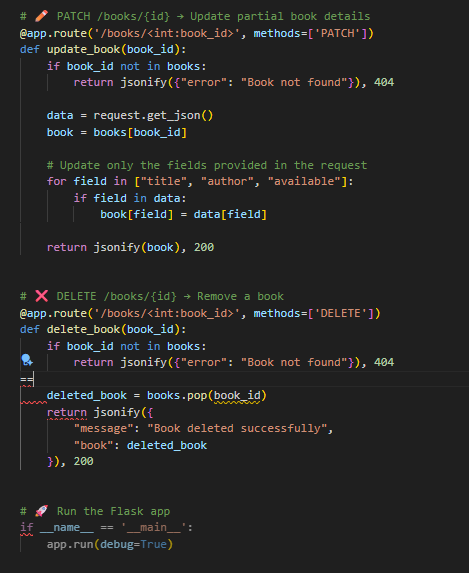
Task 2 – Library Book Management API:

Prompt: Build a complete **RESTful API using Python (Flask)** for managing student records.

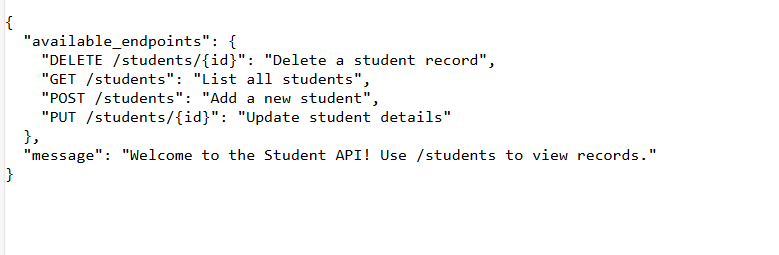
**Requirements:**

* Use an **in-memory dictionary** to store student records.
* Include these endpoints:
  + GET /students → List all students
  + POST /students → Add a new student
  + PUT /students/{id} → Update student details
  + DELETE /students/{id} → Delete a student record
* Responses must be in **JSON format**.
* Add a **home route (/)** that returns:
  + A welcome message (e.g., “Welcome to the Student API! Use /students to view records.”)
  + A list of all available endpoints and their descriptions in JSON.
* Implement **basic error handling** for invalid inputs or missing student IDs.





Output:



# Task 3 – Employee Payroll API:

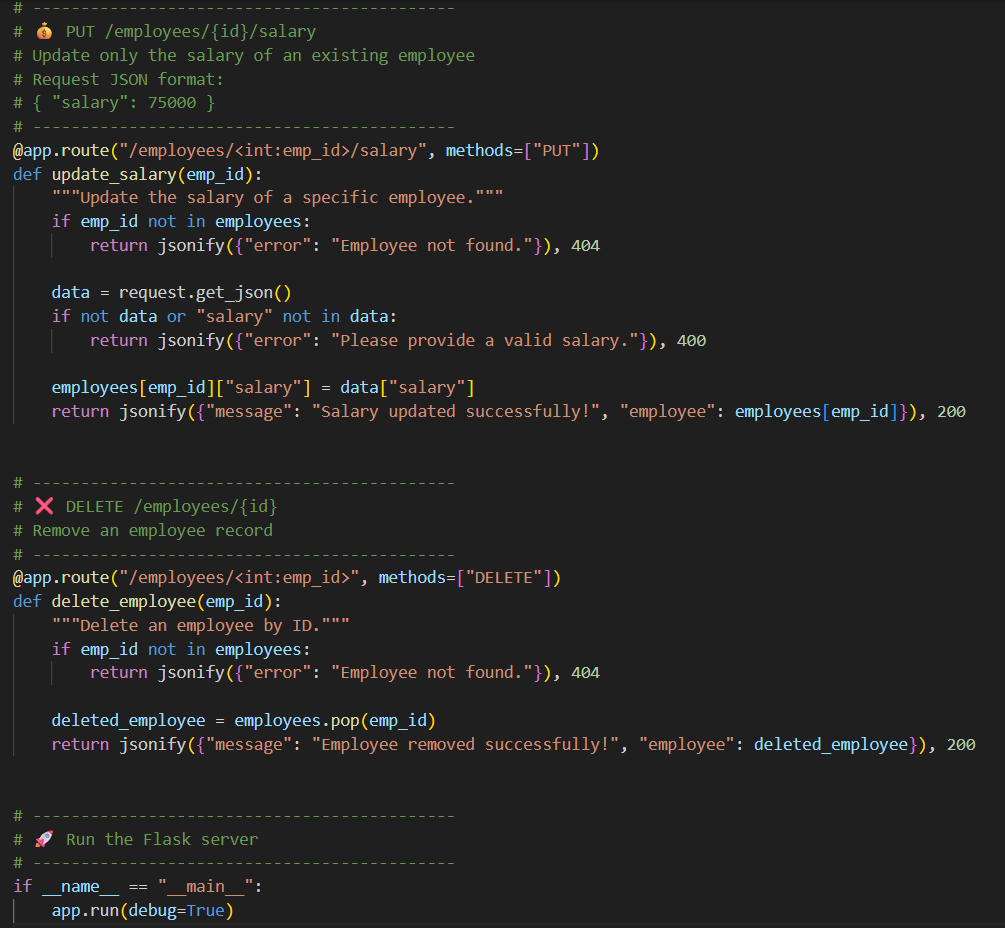
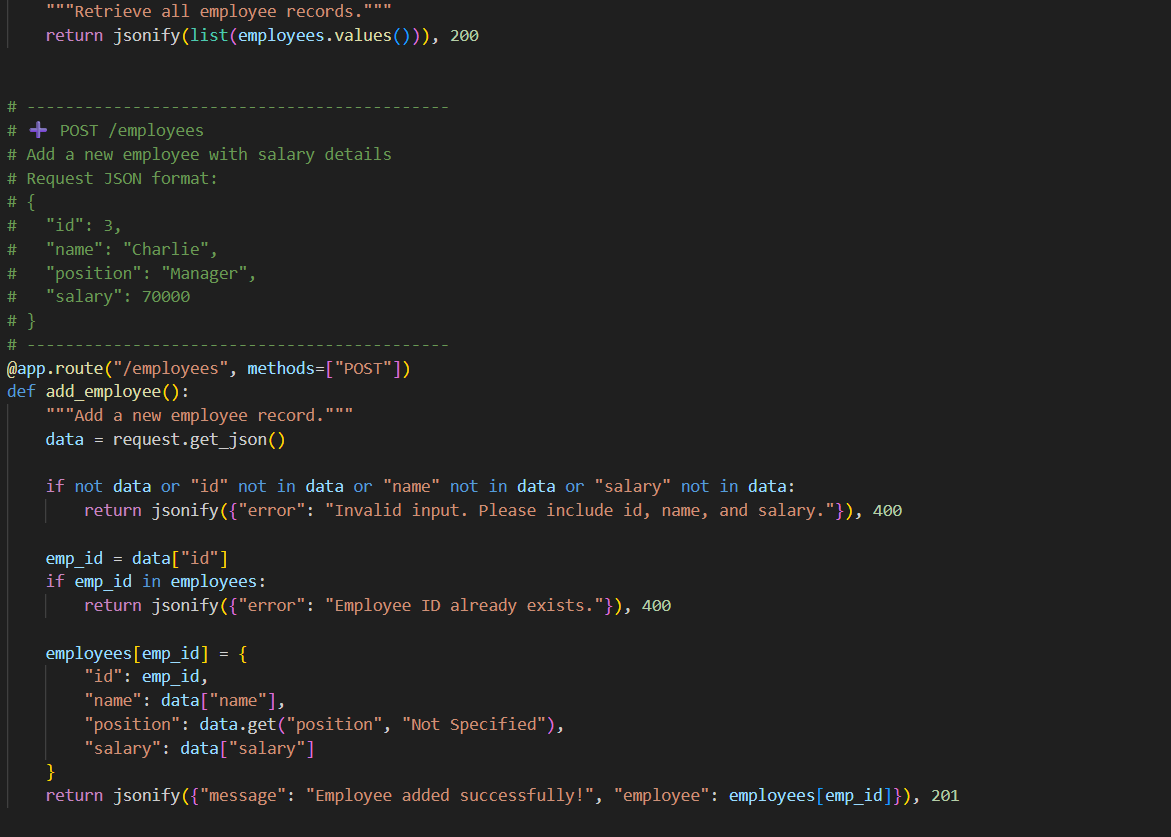
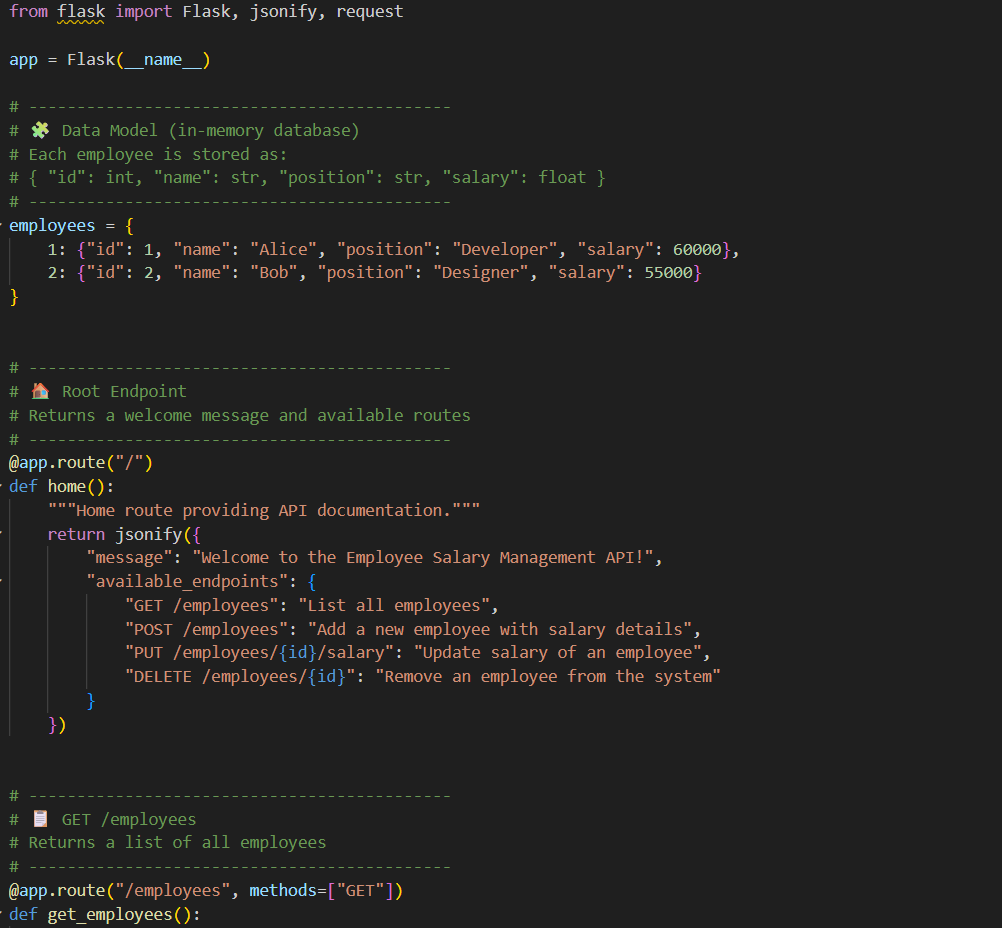
Prompt: Create a RESTful API for managing employees and their salaries using **Python Flask**

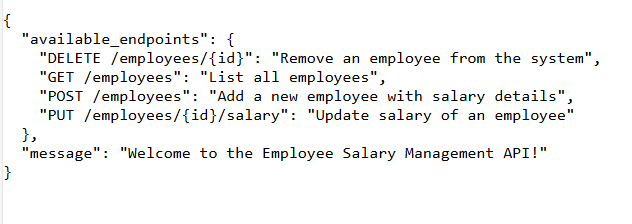
**Requirements:**

1. **Endpoints:**
   * GET /employees → List all employees
   * POST /employees → Add a new employee with salary details
   * PUT /employees/{id}/salary → Update salary of an employee
   * DELETE /employees/{id} → Remove an employee from the system
2. **Data Model Suggestion:**
   * Employee: id (unique), name, position, salary
3. **Documentation:**
   * Include comments/docstrings for each endpoint explaining its functionality.
   * Include instructions for running the API.
4. **Additional Features:**
   * Use JSON for request and response bodies
   * Proper error handling for invalid IDs or missing data

Provide complete code that is ready to run, with example requests for testing each endpoint.

Code:



Output: 

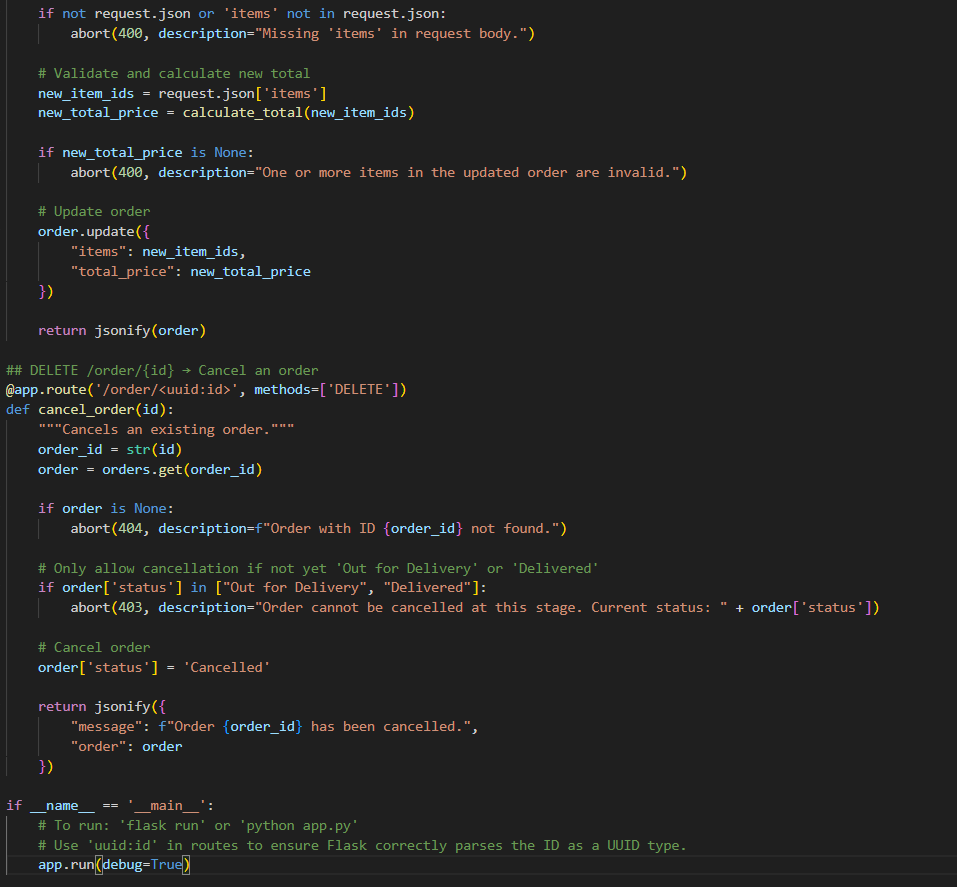
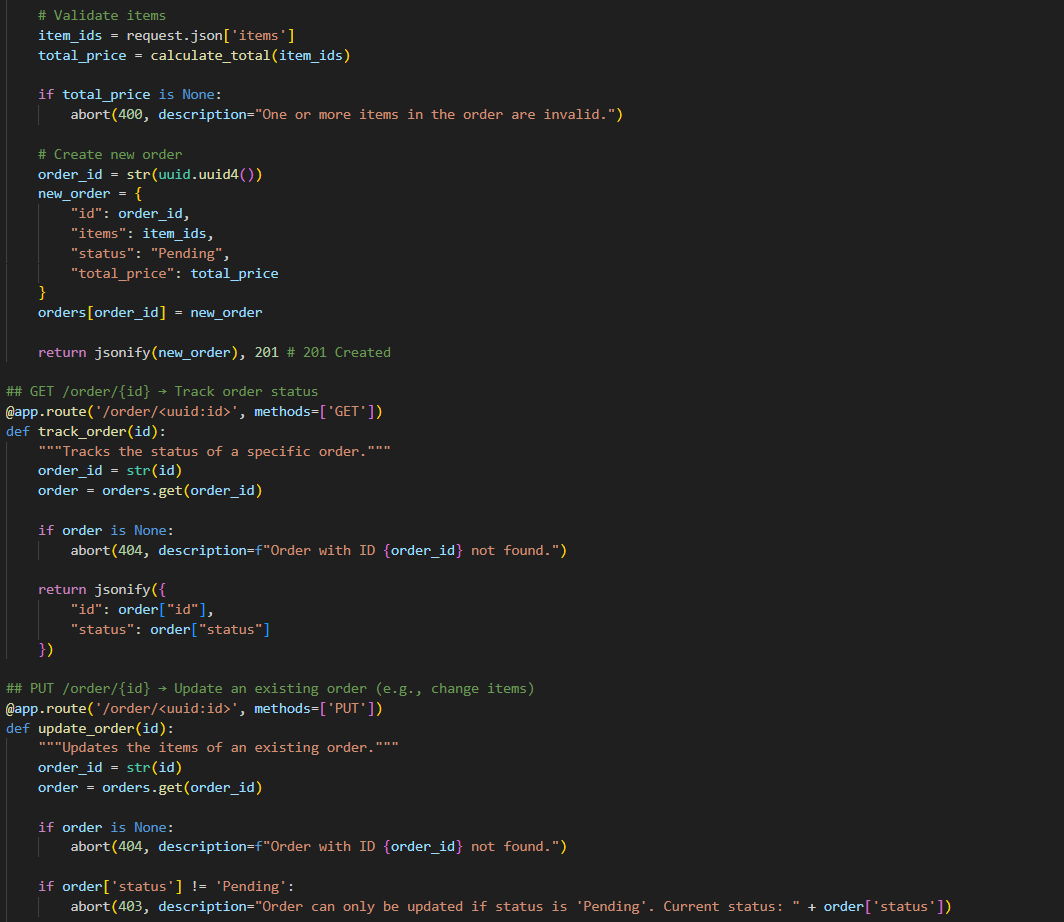
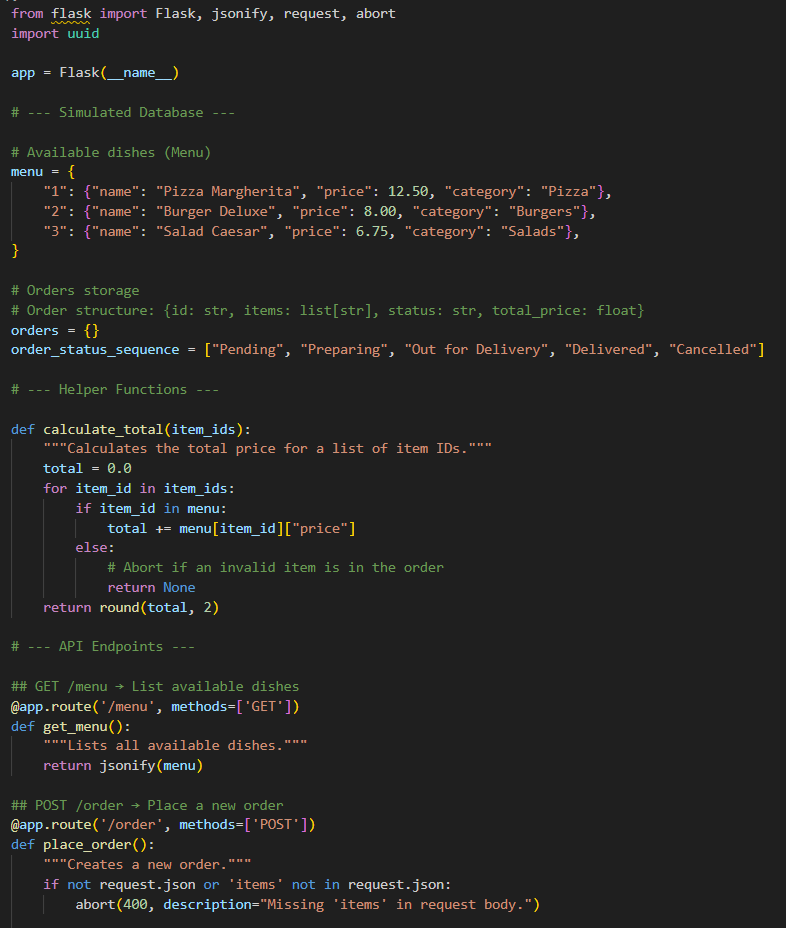
## Task 4 – Real-Time Application: Online Food Ordering API:

Design a RESTful API for an **online food ordering system** using **Node.js with Express** (or specify another framework if needed).

**Requirements:**

1. **Endpoints:**
   * GET /menu → List all available dishes
   * POST /order → Place a new order
   * GET /order/{id} → Track order status by order ID
   * PUT /order/{id} → Update an existing order (e.g., modify items)
   * DELETE /order/{id} → Cancel an order
2. **Data Models:**
   * Dish: id, name, description, price
   * Order: id, items (list of dish IDs), totalPrice, status (pending, preparing, delivered)
3. **API Features:**
   * JSON input/output for all endpoints
   * Proper error handling for invalid IDs or missing data
   * Include comments/docstrings explaining each endpoint
4. **Optional Suggested Improvements:**
   * Add authentication for placing/updating/canceling orders
   * Implement pagination for GET /menu
   * Add order history for users

Provide complete code that is ready to run, with sample requests for testing all endpoints.

Code: 

The endpoints:

