**HALL TICKET NUMBER: 2403A51329**

**NAME: B SHASHI KUMAR**

**BATCH: 24BTCAICSB13**

**AssignmentNumber:15.1**

Lab 15 – Backend API Development: Creating RESTful services  
with AI  
Lab Objectives:  
• Understand the fundamentals of RESTful API design.  
• Use AI-assisted coding tools to generate backend services.  
• Implement CRUD (Create, Read, Update, Delete) operations.  
Week8 -  
Monday

• Document endpoints with comments and auto-generated  
documentation.

Task 1 – Student Records API  
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.

**Prompt:-**

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:-**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Output:-**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Code observation:-**

1. Single-file Flask REST API (student\_api\_with\_demo.py) implementing student records stored in-memory.
2. Uses a Python list students with auto-incrementing id produced by next\_id().
3. Provides endpoints: GET /students, POST /students, PUT /students/<id>, DELETE /students/<id>, all returning JSON.
4. POST validates JSON and required fields, converts types, appends the new student, and returns 201 Created.
5. PUT does partial updates with type checks, returns the updated object or 404 if the student doesn't exist.
6. DELETE removes the record and returns a confirmation with the deleted object; 404 if not found.
7. Errors return JSON payloads and appropriate HTTP status codes (400, 404, 415, etc.).
8. A background ServerThread runs the Flask server so the script can perform demo client requests in-process.
9. The demo client issues a sequence (GET, POST, GET, PUT, DELETE, GET, health) and prints formatted JSON outputs.
10. Built for development/demo: swap to a production WSGI server and a persistent database for real deployments.

**Task 2 – Library Book Management API**

**Task:**  
Develop a RESTful API to handle library books.

**Instructions:**

* Endpoints required:
  + GET /books → Retrieve all books
  + POST /books → Add a new book
  + GET /books/{id} → Get details of a specific book
  + PATCH /books/{id} → Update partial book details (e.g., availability)
  + DELETE /books/{id} → Remove a book
* Implement error handling for invalid requests.

**Prompt:-**

Develop a RESTful API to handle library books.

**Instructions:**

* Endpoints required:
  + GET /books → Retrieve all books
  + POST /books → Add a new book
  + GET /books/{id} → Get details of a specific book
  + PATCH /books/{id} → Update partial book details (e.g., availability)
  + DELETE /books/{id} → Remove a book
* Implement error handling for invalid requests.

**Code:-**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Output:-**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Code observation:-**

1. books\_api.py is a single-file Flask REST API that manages books using an in-memory Python list.
2. Each book is a dict: id, title, author, isbn, year, available, and added\_at (ISO timestamp).
3. Helper functions next\_book\_id() and find\_book() provide auto-increment IDs and lookup by ID.
4. GET /books returns the full list of books as JSON (200 OK).
5. POST /books validates JSON payload (required fields and types), creates a new book with added\_at, and returns 201.
6. GET /books/<id> returns a single book or a JSON 404 error if not found.
7. PATCH /books/<id> accepts partial updates (title/author/isbn/year/available) with type checks and returns the updated book.
8. DELETE /books/<id> removes a book and returns the deleted record in the response (or 404 if absent).
9. The code includes consistent JSON error responses and appropriate HTTP status codes (400, 404, 415, etc.).
10. A demo runner starts the server in a background thread and issues test requests (for development only; use a WSGI server and persistent DB in production).

**Task 3 – Employee Payroll API**

**Task:**  
Create an API for managing employees and their salaries.

**Instructions:**

* Endpoints required:
  + 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 employee from system
* Use AI to:
  + Suggest data model structure.

Add comments/docstrings for all endpoints.

**Prompt:-**

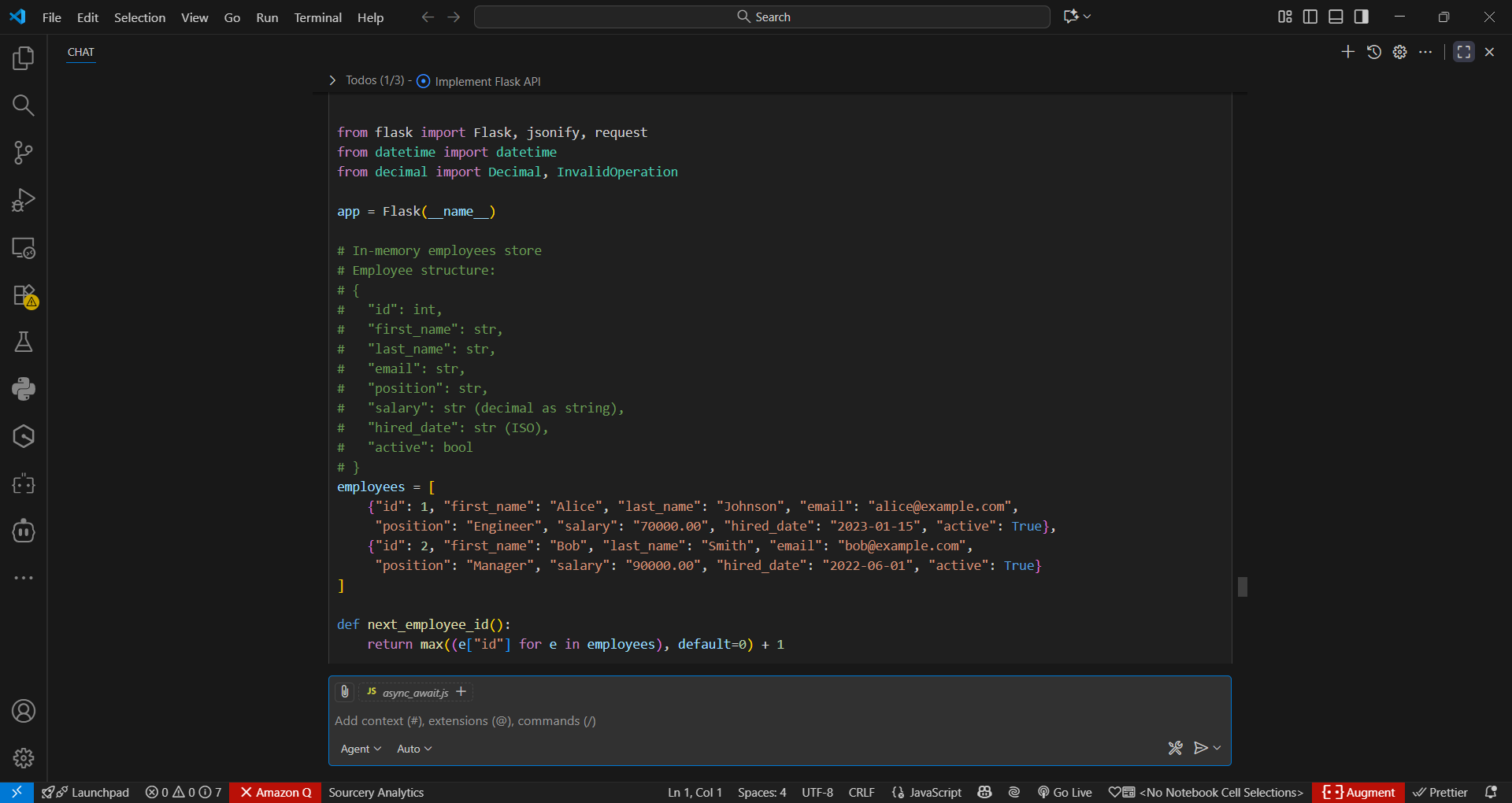
Create an API for managing employees and their salaries.

**Instructions:**

* Endpoints required:
  + 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 employee from system
* Use AI to:
  + Suggest data model structure.

Add comments/docstrings for all endpoints.

**Code:-**

****

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Output:-**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Code observation:-**

1. employee\_payroll\_api.py is a single-file Flask app implementing an Employee Payroll REST API with an in-memory list as storage.
2. Each employee record contains: id, first\_name, last\_name, email, position, salary (decimal as string), hired\_date (ISO), and active.
3. Endpoints implemented: GET /employees, POST /employees, PUT /employees/{id}/salary, DELETE /employees/{id}, plus a health GET /.
4. POST /employees requires first\_name, last\_name, email, position, salary and validates types; hired\_date defaults to today if omitted.
5. Salary validation uses Python Decimal to ensure numeric format, non-negative value, and normalizes to two decimal places.
6. PUT /employees/{id}/salary accepts a JSON salary field, validates it, updates the employee record, and returns the updated object.
7. DELETE /employees/{id} removes the employee and returns the deleted record; missing resources return JSON 404 errors.
8. All endpoints return JSON responses and proper HTTP status codes (200, 201, 400, 404, 415); docstrings describe payloads and responses.
9. Salary is stored and returned as a string to preserve exact decimal representation and avoid floating-point rounding in JSON.
10. For production use, replace in-memory storage with a database, run behind a WSGI server, add authentication, and improve input sanitization and logging.

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

**Scenario:**  
Design a simple API for an online food ordering system.

**Requirements:**

* Endpoints required:
  + GET /menu → List available dishes
  + POST /order → Place a new order
  + GET /order/{id} → Track order status
  + PUT /order/{id} → Update an existing order (e.g., change items)
  + DELETE /order/{id} → Cancel an order
* AI should generate:
  + REST API code

Suggested improvements (like authentication, pagination).

**Prompt:-**

Design a simple API for an online food ordering system.

**Requirements:**

* Endpoints required:
  + GET /menu → List available dishes
  + POST /order → Place a new order
  + GET /order/{id} → Track order status
  + PUT /order/{id} → Update an existing order (e.g., change items)
  + DELETE /order/{id} → Cancel an order
* AI should generate:
  + REST API code

Suggested improvements (like authentication, pagination).

**Code:-**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Output:-**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Code observation:-**

1. food\_ordering\_api.py is a Flask-based REST API for food ordering with in-memory storage for menu items and orders.
2. Uses Enums for order status tracking (PENDING, CONFIRMED, PREPARING, READY, DELIVERED, CANCELLED) and includes proper data validation.
3. GET /menu endpoint lists available dishes with optional filtering by category and availability status.
4. POST /order creates new orders, validating customer details and menu items, calculating totals using Python's Decimal for accuracy.
5. GET /order/{id} retrieves order details including status, items, customer info, and timestamps.
6. PUT /order/{id} allows updating pending orders with new items, recalculating totals, and tracking modification time.
7. DELETE /order/{id} implements order cancellation for pending/confirmed orders, updating status and timestamps.
8. Includes comprehensive error handling for invalid requests, missing resources, and state-dependent operations.
9. Features type hints, docstrings, and detailed comments for maintainability and documentation.
10. Includes extensive suggestions for production improvements like authentication, database storage, payment integration, and monitoring.