# Lab Assignment-15

## Task 1

**Student Records API**

**Task:**

Use AI to build a RESTful API for managing student records.

**Instructions:**

- Endpoints required:
  - GET /students → List all students
  - POST /students → Add a new student
  - PUT /students/{id} → Update student details
  - 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.

```
# Install dependencies
!pip install flask flask-ngrok > /dev/null 2>&1

# ----------------------------
# Student Records API
# ----------------------------
from flask import Flask, jsonify, request
from flask_ngrok import run_with_ngrok

app = Flask(__name__)
run_with_ngrok(app)  # Automatically run ngrok when app starts

# In-memory data structure (Dictionary)
students = {
    1: {"name": "Alice", "age": 20, "course": "Computer Science"},
    2: {"name": "Bob", "age": 22, "course": "Mechanical Engineering"}
}

# ----------------------------
# GET /students - List all students
```

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

# ----------------------------
# POST /students - Add a new student
# ----------------------------
@app.route('/students', methods=['POST'])
def add_student():
    data = request.get_json()
    if not data or not all(k in data for k in ("name", "age", "course")):
        return jsonify({"error": "Missing fields (name, age, course required)"}), 400

    new_id = max(students.keys(), default=0) + 1
    students[new_id] = {
        "name": data["name"],
        "age": data["age"],
        "course": data["course"]
```

[14] ✓ 5m

```python
            "course": data["course"]
        }
        return jsonify({"message": "Student added successfully", "student": students[new_id]}), 201

    # ----------------------------
    # PUT /students/<id> - Update student details
    # ----------------------------
    @app.route('/students/<int:student_id>', methods=['PUT'])
    def update_student(student_id):
        if student_id not in students:
            return jsonify({"error": "Student not found"}), 404

        data = request.get_json()
        students[student_id].update(data)
        return jsonify({"message": "Student updated successfully", "student": students[student_id]}), 200

    # ----------------------------
    # DELETE /students/<id> - Delete a student record
    # ----------------------------
    @app.route('/students/<int:student_id>', methods=['DELETE'])
```

```python
    data = request.get_json()
    students[student_id].update(data)
    return jsonify({"message": "Student updated successfully", "student": students[student_id]}), 200

# ----------------------------
# DELETE /students/<id> - Delete a student record
# ----------------------------
@app.route('/students/<int:student_id>', methods=['DELETE'])
def delete_student(student_id):
    if student_id not in students:
        return jsonify({"error": "Student not found"}), 404

    del students[student_id]
    return jsonify({"message": "Student deleted successfully"}), 200

# ----------------------------
# Run Flask app with ngrok
# ----------------------------
app.run()
```

# 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.

**Expected Output:**

- Functional API with CRUD + partial updates.

```
# Install required packages
!pip install flask pyngrok > /dev/null 2>&1

from flask import Flask, jsonify, request
from pyngrok import ngrok

# ----------------------------
# Library Book Management API
# ----------------------------
app = Flask(__name__)

# In-memory data (sample)
books = {
    1: {"title": "1984", "author": "George Orwell", "available": True},
    2: {"title": "To Kill a Mockingbird", "author": "Harper Lee", "available": False},
}

# ----------------------------
# GET /books - Retrieve all books
```

```python
@app.route('/books', methods=['GET'])
def get_books():
    return jsonify({
        "status": "success",
        "books": books
    }), 200

# ----------------------------
# POST /books - Add a new book
# ----------------------------
@app.route('/books', methods=['POST'])
def add_book():
    data = request.get_json()
    if not data or not all(k in data for k in ("title", "author", "available")):
        return jsonify({
            "status": "error",
            "message": "Missing fields: title, author, and available are required"
        }), 400

    new_id = max(books.keys(), default=0) + 1
```

```python
  new_id = max(books.keys(), default=0) + 1
  books[new_id] = {
      "title": data["title"],
      "author": data["author"],
      "available": data["available"]
  }

  return jsonify({
      "status": "success",
      "message": "Book added successfully",
      "book": {new_id: books[new_id]}
  }), 201

----------------------------
GET /books/<id> - Retrieve a specific book
----------------------------
app.route('/books/<int:book_id>', methods=['GET'])
ef get_book(book_id):
  if book_id not in books:
```

```python
def get_book(book_id):
    if book_id not in books:
        return jsonify({
            "status": "error",
            "message": "Book not found"
        }), 404

    return jsonify({
        "status": "success",
        "book": {book_id: books[book_id]}
    }), 200

# ----------------------------
# PATCH /books/<id> - Partial update (e.g., availability)
# ----------------------------
@app.route('/books/<int:book_id>', methods=['PATCH'])
def update_book_partial(book_id):
    if book_id not in books:
        return jsonify({
            "status": "error",
```

```python
     return jsonify({
         "status": "error",
         "message": "Book not found"
     }), 404

 data = request.get_json()
 if not data:
     return jsonify({
         "status": "error",
         "message": "No data provided for update"
     }), 400

 # Only update existing keys
 for key, value in data.items():
     if key in books[book_id]:
         books[book_id][key] = value

 return jsonify({
     "status": "success",
     "message": "Book updated successfully",
```

```
    }), 200

# ----------------------------
# DELETE /books/<id> - Remove a book
# ----------------------------
@app.route('/books/<int:book_id>', methods=['DELETE'])
def delete_book(book_id):
    if book_id not in books:
        return jsonify({
            "status": "error",
            "message": "Book not found"
        }), 404

    del books[book_id]
    return jsonify({
        "status": "success",
        "message": "Book deleted successfully"
    }), 200

# ----------------------------
```

```
    if book_id not in books:
        return jsonify({
            "status": "error",
            "message": "Book not found"
        }), 404

    del books[book_id]
    return jsonify({
        "status": "success",
        "message": "Book deleted successfully"
    }), 200

# ----------------------------
# Run Flask app via ngrok
# ----------------------------
public_url = ngrok.connect(5000)
print(f"✅ Library API is live at: {public_url}")
print("Use this URL in Postman or browser to test endpoints.")
app.run(port=5000)
```

# 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.

**Expected Output:**

- API supporting salary management with clear documentation.

```
# Install lightweight Colab-friendly package that auto-exposes Flask API
!pip install flask-ngrok-free > /dev/null 2>&1

from flask import Flask, jsonify, request
from flask_ngrok_free import run_app

# ------------------------------------------------------------
# 💼 Employee Payroll API
# ------------------------------------------------------------
app = Flask(__name__)

# ------------------------------------------------------------
# 🧠 Suggested Data Model (In-memory dictionary)
# ------------------------------------------------------------
# Each employee record is structured as:
# employees = {
#   emp_id: {
#       "name": str,
#       "department": str,
#       "position": str,
#       "salary": float
#   }
# }
employees = {
    1: {"name": "Alice", "department": "HR", "position": "Manager", "salary": 60000.0},
    2: {"name": "Bob", "department": "IT", "position": "Developer", "salary": 55000.0},
}

# ------------------------------------------------------------
# GET /employees → List all employees
# ------------------------------------------------------------
@app.route('/employees', methods=['GET'])
def get_employees():
    """
    Retrieve a list of all employees and their details.
    Returns:
        JSON: List of all employee records with IDs.
    """
    return jsonify({
        "status": "success",
```

```
        "status": "success",
        "employees": employees
    }), 200

# ------------------------------------------------------------
# POST /employees → Add a new employee with salary details
# ------------------------------------------------------------
@app.route('/employees', methods=['POST'])
def add_employee():
    """
    Add a new employee to the payroll system.
    Expected JSON body:
        {
            "name": "John",
            "department": "Finance",
            "position": "Analyst",
            "salary": 50000
        }
    Returns:
        JSON: Success message with created employee details.
```

```
        return jsonify({
            "status": "error",
            "message": "Employee not found"
        }), 404

    data = request.get_json()
    if not data or "salary" not in data:
        return jsonify({
            "status": "error",
            "message": "Salary field required"
        }), 400

    employees[emp_id]["salary"] = float(data["salary"])

    return jsonify({
        "status": "success",
        "message": "Salary updated successfully",
        "employee": {emp_id: employees[emp_id]}
    }), 200

# ------------------------------------------------------------
```

```
        "employee": {emp_id: employees[emp_id]}
    }), 200

# ------------------------------------------------------------
# DELETE /employees/<id> → Remove employee from system
# ------------------------------------------------------------
@app.route('/employees/<int:emp_id>', methods=['DELETE'])
def delete_employee(emp_id):
    """
    Delete an employee record from the system.
    URL Parameter:
        emp_id (int): Employee ID
    Returns:
        JSON: Success message if deleted, error if not found.
    """
    if emp_id not in employees:
        return jsonify({
            "status": "error",
            "message": "Employee not found"
        }), 404
```

```
    """
    if emp_id not in employees:
        return jsonify({
            "status": "error",
            "message": "Employee not found"
        }), 404

    del employees[emp_id]

    return jsonify({
        "status": "success",
        "message": "Employee deleted successfully"
    }), 200

# ------------------------------------------------------------
# Run Flask app in Colab (auto-public URL)
# ------------------------------------------------------------
print("✅ Employee Payroll API is running... use the public URL below 👇")
run_app(app, port=5000)
```

# 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)

**Expected Output:**

- Fully working API simulating a food ordering backend.

---

```
# Install lightweight library that lets Flask run publicly in Colab
!pip install flask-ngrok-free > /dev/null 2>&1

from flask import Flask, jsonify, request
from flask_ngrok_free import run_app
import time

# ------------------------------------------------------------
# 🍽️ Online Food Ordering API
# ------------------------------------------------------------
app = Flask(__name__)

# ------------------------------------------------------------
# 🧠 Suggested Data Model (In-memory)
# ------------------------------------------------------------
# menu = { item_id: {"name": str, "price": float, "available": bool} }
# orders = { order_id: {"items": [item_ids], "status": str, "timestamp": str} }

menu = {
```

```python
menu = {
    1: {"name": "Margherita Pizza", "price": 250.0, "available": True},
    2: {"name": "Paneer Burger", "price": 150.0, "available": True},
    3: {"name": "Veg Biryani", "price": 180.0, "available": True},
    4: {"name": "Cold Coffee", "price": 90.0, "available": True},
}

orders = {}
order_counter = 0

# ------------------------------------------------------------
# GET /menu → List available dishes
# ------------------------------------------------------------
@app.route('/menu', methods=['GET'])
def get_menu():
    """
    Retrieve all available dishes in the menu.
    Returns:
        JSON list of dishes with name, price, and availability.
    """
    """
    return jsonify({
        "status": "success",
        "menu": menu
    }), 200

# ------------------------------------------------------------
# POST /order → Place a new order
# ------------------------------------------------------------
@app.route('/order', methods=['POST'])
def place_order():
    """
    Place a new food order.
    Expected JSON body:
        { "items": [1, 3, 4] }
    Returns:
        JSON confirmation with order_id and order details.
    """
    global order_counter
    data = request.get_json()
```

```
    if not data or "items" not in data:
        return jsonify({
            "status": "error",
            "message": "Missing 'items' in request body."
        }), 400

    # Validate that all items exist
    invalid_items = [i for i in data["items"] if i not in menu or not menu[i]["available"]]
    if invalid_items:
        return jsonify({
            "status": "error",
            "message": f"Invalid or unavailable items: {invalid_items}"
        }), 400

    # Create new order
    order_counter += 1
    orders[order_counter] = {
        "items": data["items"],
        "status": "Order Placed",
    }

    return jsonify({
        "status": "success",
        "message": "Order placed successfully!",
        "order_id": order_counter,
        "order_details": orders[order_counter]
    }), 201

# ---------------------------------------------------------------
# GET /order/<id> → Track order status
# ---------------------------------------------------------------
@app.route('/order/<int:order_id>', methods=['GET'])
def get_order(order_id):
    """
    Get details and status of a specific order.
    URL Parameter:
        order_id (int)
    Returns:
        JSON order details or error if not found.
    """
```

```
    if order_id not in orders:
        return jsonify({
            "status": "error",
            "message": "Order not found."
        }), 404

    return jsonify({
        "status": "success",
        "order_id": order_id,
        "order_details": orders[order_id]
    }), 200

# ------------------------------------------------------------
# PUT /order/<id> → Update an existing order (e.g., change items)
# ------------------------------------------------------------
@app.route('/order/<int:order_id>', methods=['PUT'])
def update_order(order_id):
    """
    Update an existing order by changing its items.
    Expected JSON body:

  invalid_items = [i for i in data["items"] if i not in menu or not menu[i]["available"]]
  if invalid_items:
      return jsonify({
          "status": "error",
          "message": f"Invalid or unavailable items: {invalid_items}"
      }), 400

  orders[order_id]["items"] = data["items"]
  orders[order_id]["status"] = "Order Updated"
  orders[order_id]["timestamp"] = time.strftime("%Y-%m-%d %H:%M:%S")

  return jsonify({
      "status": "success",
      "message": "Order updated successfully.",
      "order_id": order_id,
      "order_details": orders[order_id]
  }), 200

# ------------------------------------------------------------
# DELETE /order/<id> → Cancel an order
```

```
    """
    if order_id not in orders:
        return jsonify({
            "status": "error",
            "message": "Order not found."
        }), 404

    del orders[order_id]

    return jsonify({
        "status": "success",
        "message": "Order cancelled successfully."
    }), 200

# ------------------------------------------------------------
# Run Flask app publicly in Colab
# ------------------------------------------------------------
print("✅ Online Food Ordering API is live! Use the public URL below 👇")
run_app(app, port=5000)
```