Worksheet 3: Develop a RESTful CRUD API for Students using FastAPI and HTML Frontend

Course: IIT67-272 | Fundamentals of Backend Architecture
Topic: RESTful API with Frontend Integration
Duration: 3 Hours
Tools: Python, FastAPI, Uvicorn, SQLite, HTML, JavaScript (Fetch API)

🎯 Learning Objectives

By the end of this lab, students will be able to:

1. Implement a **RESTful API** with FastAPI supporting CRUD operations.
2. Validate and manage student data using **Pydantic and SQLAlchemy**.
3. Create a simple **frontend interface (HTML + JS)** to call backend APIs.
4. Understand **client–server communication** and API response handling.

🧠 Concept Recap

- **FastAPI** provides the backend (server) handling requests and responses.
- **Frontend (HTML + JavaScript)** sends requests via **Fetch API** and displays data.
- Communication uses **JSON** over HTTP (RESTful architecture).

🧩 Task Description

You will build:

1. A **FastAPI server** exposing endpoints:
    - `GET /students` — Retrieve all students
    - `POST /students` — Add a new student
    - `PUT /students/{id}` — Update student info
    - `DELETE /students/{id}` — Remove student
2. A **Frontend HTML page** that:
    - Displays all students
    - Allows adding, editing, and deleting students dynamically

💻 Project Structure

```
fastapi student/
 ├── main.py
 ├── models.py
 ├── schemas.py
```

```
├── database.py
├── static/
│   └── index.html
└── requirements.txt
```

⚙️ Backend (FastAPI)

Step 1: Install dependencies

```
pip install fastapi uvicorn sqlalchemy pydantic
```

Step 2: Database & Models

```
# database.py
from sqlalchemy import create_engine, MetaData
engine = create_engine("sqlite:///./students.db")
meta = MetaData()
```

```
# models.py
from sqlalchemy import Table, Column, Integer, String
from database import meta, engine

students = Table(
    "students", meta,
    Column("id", Integer, primary_key=True),
    Column("name", String(50)),
    Column("email", String(50)),
    Column("major", String(50))
)
meta.create_all(engine)
```

Step 3: Pydantic Schema

```
# schemas.py
from pydantic import BaseModel, EmailStr

class Student(BaseModel):
    name: str
    email: EmailStr
    major: str
```

Step 4: FastAPI Main Application

```
# main.py
from fastapi import FastAPI, HTTPException
from fastapi.staticfiles import StaticFiles
from fastapi.middleware.cors import CORSMiddleware
```

```
from schemas import Student
from models import students
from database import engine
from sqlalchemy import select

app = FastAPI(title="Student Management API")

# Serve static frontend
app.mount("/", StaticFiles(directory="static", html=True),

# CORS for frontend access
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

@app.get("/students")
def get_students():
    with engine.connect() as conn:
        result = conn.execute(select(students)).fetchall()
        return [dict(r) for r in result]

@app.post("/students")
def create_student(student: Student):
    with engine.connect() as conn:
        conn.execute(students.insert().values(**student.dic
        conn.commit()
        return {"message": "Student added successfully"}

@app.put("/students/{id}")
def update_student(id: int, student: Student):
    with engine.connect() as conn:
        conn.execute(students.update().where(students.c.id
        conn.commit()
        return {"message": "Student updated"}

@app.delete("/students/{id}")
def delete_student(id: int):
    with engine.connect() as conn:
        conn.execute(students.delete().where(students.c.id
        conn.commit()
        return {"message": "Student deleted"}
```

Run the server

```
uvicorn main:app --reload
```

🌐 Frontend (HTML + JavaScript)

File: static/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Student Management System</title>
  <style>
    body { font-family: Arial; margin: 2em; color: #333; }
    table { width: 100%; border-collapse: collapse; margin-
    th, td { border: 1px solid #aaa; padding: 8px; text-ali
    th { background-color: #f0f0f0; }
    input { margin: 5px; padding: 6px; }
    button { padding: 6px 10px; margin: 4px; }
  </style>
</head>
<body>
  <h1>🎓 Student Management</h1>

  <div>
    <input id="name" placeholder="Name">
    <input id="email" placeholder="Email">
    <input id="major" placeholder="Major">
    <button onclick="addStudent()">Add Student</button>
  </div>

  <table id="studentTable">
    <thead>
      <tr><th>ID</th><th>Name</th><th>Email</th><th>Major</
    </thead>
    <tbody></tbody>
  </table>

  <script>
    const API_URL = "/students";

    async function loadStudents() {
      const res = await fetch(API_URL);
      const data = await res.json();
      const tbody = document.querySelector("#studentTable t
      tbody.innerHTML = "";
      data.forEach(stu => {
        const row = `
          <tr>
            <td>${stu.id}</td>
            <td>${stu.name}</td>
            <td>${stu.email}</td>
            <td>${stu.major}</td>
            <td>
              <button onclick="deleteStudent(${stu.id})">🗑
            </td>
          </tr>`;
        tbody.innerHTML += row;
      });
```

```
    }

    async function addStudent() {
      const name = document.getElementById("name").value;
      const email = document.getElementById("email").value;
      const major = document.getElementById("major").value;

      await fetch(API_URL, {
        method: "POST",
        headers: {"Content-Type": "application/json"},
        body: JSON.stringify({name, email, major})
      });
      loadStudents();
    }

    async function deleteStudent(id) {
      await fetch(`${API_URL}/${id}`, { method: "DELETE" })
      loadStudents();
    }

    loadStudents();
  </script>
</body>
</html>
```

🧪 Test Plan

1. Run the server → `http://127.0.0.1:8000`
2. Open the browser and load `index.html`.
3. Perform the following actions:
    - Add new students
    - View all students in the table
    - Delete students
4. Observe JSON responses and verify database updates.

📋 Evaluation Criteria (100 pts)

Component	Points
FastAPI CRUD implementation	30
API validation (Pydantic)	10
Database connection & schema	15
HTML interface functionality	25
Fetch API integration & JSON handling	15
Code quality & comments	5

🧭 Reflection Questions

- How does the frontend communicate with the backend using Fetch API?
- What happens if the API returns an error (e.g., 404 or 500)?
- Why do we separate concerns between backend (FastAPI) and frontend (HTML/JS)?

References

- FastAPI Documentation
- Fetch API Guide – MDN
- SQLAlchemy ORM
- Pydantic Models