Lab Sheet 02 : Backend Architecture & RESTful API Development using FastAPI

Based on Chapter 2: *Fundamentals of Backend Architecture*

Learning Objectives

After completing this lab, students will be able to:

- Explain backend architectural concepts: Client–Server, REST, Layered, MVC
- Create RESTful APIs using **FastAPI**
- Organize project code following **Layered Architecture**
- Apply **Dependency Injection** and **Pydantic models**
- Test API endpoints using **Swagger UI / Postman**

Pre-Lab: Concept Review

Instructions: Review the following concepts and answer briefly.

1. What are the roles of **Client** and **Server** in backend architecture?
   Your answer:
   ……
   ……
2. Why is REST called a **stateless** architecture?
   Your answer:
   ……
   ……
3. Describe how the **Layered Architecture** helps developers organize code.
   Your answer:
   ……
   ……

Lab 2.1 — Exploring Client–Server and REST API

Objective

Understand how RESTful communication works between client and server.

Tasks

1. Use **Postman** or `curl` to send:

```
GET https://jsonplaceholder.typicode.com/users/1
```

2. Observe and record:

- HTTP method used
- Status code
- Response headers
- JSON body

3. Draw a simple diagram showing the **Client–Server** communication flow.

Your diagram / explanation:
……

……
……

Lab 2.2 — Building Your First FastAPI App

Objective

Set up FastAPI and create a simple REST endpoint.

Setup

Install required packages:

```
pip install fastapi uvicorn
```

Code Example

```
# main.py
from fastapi import FastAPI

app = FastAPI()

@app.get("/hello")
def say_hello():
    return {"message": "Hello, Backend!"}
```

Run:

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

Open in browser: http://127.0.0.1:8000/docs

Tasks

- Try calling `/hello` endpoint in Swagger UI.
- Note the request and response shown.

Screenshot / Observation: ……

Lab 2.3 — Applying Layered Architecture

Objective

Organize code into Controller, Service, and Repository.

Folder Structure

```
project/
    main.py
    services/
        student_service.py
    repositories/
        student_repo.py
```

student_repo.py

```
students = [{"id": 1, "name": "Ananya", "major": "IT"}]

def get_students():
    return students
```

student_service.py

```
from repositories.student_repo import get_students

def list_students():
    return get_students()
```

main.py

```
from fastapi import FastAPI
from services.student_service import list_students

app = FastAPI()

@app.get("/students")
def get_students():
    return list_students()
```

Questions

1. Which part acts as the **Controller**? ……
2. Which handles **Business Logic**? ……
3. Which interacts with data? ……

Lab 2.4 — Designing RESTful Endpoints

Objective

Create multiple endpoints using HTTP methods (GET, POST).

Code Example

```
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Student(BaseModel):
    id: int
    name: str
    major: str

students = []

@app.get("/api/students")
def get_all():
    return students

@app.post("/api/students")
def add_student(s: Student):
    students.append(s)
    return s
```

Tasks

1. Use Swagger UI to add new students.
2. Retrieve all students with `GET /api/students`.
3. Observe the returned JSON list.

Observation: …… ……

Lab 2.5 — Connecting MVC & Database (SQLModel + SQLite)

Objective

Integrate FastAPI with an in-memory database using SQLModel.

Install:

```
pip install sqlmodel
```

student_model.py

```python
from sqlmodel import SQLModel, Field

class Student(SQLModel, table=True):
    id: int | None = Field(default=None, primary_key=True)
    name: str
    major: str
```

main.py

```python
from fastapi import FastAPI, Depends
from sqlmodel import SQLModel, Session, create_engine, select
from models.student_model import Student

engine = create_engine("sqlite:///database.db")
app = FastAPI()

@app.on_event("startup")
def on_startup():
    SQLModel.metadata.create_all(engine)

def get_session():
    with Session(engine) as session:
        yield session

@app.get("/students")
def get_students(session: Session = Depends(get_session)):
    return session.exec(select(Student)).all()
```

Questions

1. Which component represents the **Model** in MVC? ……
2. What is the role of the database engine here? ……
3. Why is Dependency Injection (`Depends`) useful? ……

Mini Project: Build Your Own API

Task

Design a small REST API using FastAPI.

Example ideas:

- Book Management
- To-Do List
- Student Records

Requirements

- At least 3 endpoints (`GET`, `POST`, `DELETE`)
- Use Pydantic Model for data validation
- Follow Layered structure
- Include one diagram showing your architecture

Project Name: …… *Endpoints:* …… *Diagram (attach or draw below):* ……

Post-Lab Reflection

1. What challenges did you face in organizing backend code? ……
   ……
2. How does FastAPI help simplify backend development? ……
   ……
3. Which part of the Layered Architecture (Controller, Service, Repository) did you find most important to maintain system clarity? ……

Summary

Concept	Tool / Practice	Skill Developed
Client–Server	curl / Postman	Communication flow
REST	Endpoint design	HTTP Methods
Layered Architecture	FastAPI modules	Modularity
MVC	Model + Controller	Code structure
DI & Framework	FastAPI + SQLModel	Scalable backend

Tip: Use this lab as your foundation for the final project — extend your API with real database, authentication, or CRUD operations.