INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

S25CS6.201
Introduction to Software Systems

FASTAPI: INTEGRATION CONNECT WITH DB AND FRONTEND

16 April 2025

Quick Recap

- What is FastAPI?
- Decorators
- Routers
- Pydantic Models
- Swagger Docs
- Jinja2 Templates Basics
- Static Files
- Directory Structure

What is FastAPI?

- FastAPI is a modern, fast web framework for building APIs with Python 3.6+
- Supports asynchronous programming (async/await)
- Built on top of Starlette and Pydantic
- Used by Uber, Netflix, Microsoft
- Auto-generates Swagger Docs

Key Concepts

- **Decorators:** Special syntax to bind routes (e.g., @app.get("/"))
- **Routers:** Modularize routes
- **Pydantic Models:** Data validation using Python types
- **Swagger Docs:** Auto-generated interactive API documentation
- **Jinja Templates:** Render HTML with embedded Python logic

Jinja Basics

- Use {{ variable }} to display data
- Use {% for %} and {% if %} for logic
- Create base.html and extend using {% block content %}
- Static files: mount CSS/JS/images via app.mount()

SQL with FastAPI - Intro

- SQLModel = SQLAlchemy + Pydantic
- Best for structured data (students, users, products)
- We'll use SQLite for simplicity

SQL – Setup

SQLite Setup:

```
pip install sqlmodel sqlite3
```

MySQL Setup:

```
pip install sqlmodel pymysql
```

File structure:

```
- main.py
- models.py
- templates/
```

What is a Model?

- A model is a Python class that defines the structure of your data.
- In FastAPI with SQLModel, models are used to create database tables.
- Each attribute in the class becomes a column in the table.

Define a Model:

```
from sqlmodel import SQLModel, Field

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

Create and Connect DB

For SQLite:

```
from sqlmodel import Session, create_engine

db_url = "sqlite:///students.db"
engine = create_engine(db_url, echo=True)

SQLModel.metadata.create_all(engine)
```

For MySQL:

```
from sqlmodel import Session, create_engine

db_url = "mysql+pymysql://username:password@localhost:3306/dbname"
engine = create_engine(db_url, echo=True)

SQLModel.metadata.create_all(engine)
```

SQL – CRUD Operations

```
@app.post("/add")
def add_student(name: str, email: str):
    with Session(engine) as session:
        student = Student(name=name, email=email)
        session.add(student)
        session.commit()
```

Render with Jinja

```
from fastapi.templating import Jinja2Templates
from fastapi.requests import Request

templates = Jinja2Templates(directory="templates")

@app.get("/students")
def show_students(request: Request):
    with Session(engine) as session:
        students = session.query(Student).all()
    return templates.TemplateResponse("students.html”,
"request": request, "students": students})
```

MongoDB – Recap

- MongoDB is a NoSQL database (stores data as JSON-like documents)
- Good for unstructured data (logs, posts, feedback)
- We'll use Motor, an async driver

MongoDB – Setup

```
pip install motor
```

Connect to Mongo:

```
from motor.motor_asyncio import AsyncIOMotorClient
client = AsyncIOMotorClient("mongodb://localhost:27017")
db = client.labdb
```

MongoDB – Insert and Find

```
@app.post("/add_post")
async def add_post(title: str, content: str):
    await db.posts.insert_one({"title": title, "content":
content})

@app.get("/posts")
async def show_posts(request: Request):
    posts = await db.posts.find().to_list(100)
    return templates.TemplateResponse("posts.html",
{"request": request, "posts": posts})
```

MongoDB – HTML Display

```
<!-- posts.html -->
{% for post in posts %}
  <h3>{{ post.title }}</h3>
  <p>{{ post.content }}</p>
{% endfor %}
```

Combining SQL and MongoDB

- SQL: Store structured user or admin data
- MongoDB: Store logs, feedback, posts
- Access both in the same FastAPI project

```
pip install sqlmodel pymysql

pip install sqlmodel sqlite3

pip install motor
```

Some Jinja Tips

- form action="/submit" method="post"
- Use request.form[] to capture POST data
- Use {% include %}, {% extends %} to modularize templates
- Use {{ get_flashed_messages() }} with Starlette for alerts

Conclusion

- FastAPI works well with both SQL and MongoDB
- Jinja templates make it easy to render data
- Practice CRUD + Templates for both DBs

Time for the Activity