**Fitness Studio App**

**Description:**  
Designed and implemented a RESTful Booking API for a fictional fitness studio offering Yoga, Zumba, and HIIT classes. The API allows clients to view available classes, book sessions, and check their bookings.

**Tech Stack:** Python, FastAPI, Pydantic, Uvicorn, Pytest

**Task that performed:**

### Core Functional Tasks

* GET /classes – Returns all available classes with name, date/time, instructor, and available slots.
* POST /book – Allows users to book a class by class ID. Validates availability and records booking.
* GET /bookings?email=... – Fetches all bookings made by a specific email address.

### Internal Tasks

* In-memory data storage using lists to simulate a database
* Datetime management for class scheduling and booking timestamps
* Validation and error handling using FastAPI and Pydantic
* Modular architecture with separate files for models, schemas, CRUD, and main app
* Automatic Swagger UI documentation for easy testing
* Unit tests for endpoints using Pytest and FastAPI's TestClient

**Libraries Used and Their Purpose:**

**1.FastAPI Library:**

* FastAPI is a modern high-performant web framework for building APIs with Python.
* It is designed to make it easy to build APIs quickly and efficiently while providing features like automatic validation, serialization, and documentation of your API.
* In this, we used Modules like FastAPI, HTTPException, Query.

**Components of Library ,**

• fastapi.FastAPI: Creates the FastAPI application instance.

• fastapi.HTTPException: Allows raising custom HTTP error responses

• fastapi.Query: Used for query parameter validation

**2.Pydantic Library:**

• Pydantic is a Python library that helps us in defining and validating data models easily.

* Key features are Type Validation, Data Parsing, Error Handling, Field Validation

• In this module we use BaseModel, EmailStr.

**Components of Library ,**

* pydantic.BaseModel: Base class for defining FastAPI-compatible data models.

• pydantic.EmailStr: Type that validates email formats automatically.

**3.Typing Library:**

**•** typing.List: Type hint for functions returning a list of objects.

**4.DateTime Library:**

• datetime: Provides datetime.now() and timedelta for setting class schedules

• timedelta: Used to create classes in the future

**Project Structure:**

* **fitness\_booking/**
* **├── app/**
* **│ ├── main.py # FastAPI app and routes**
* **│ ├── models.py # Data models (in-memory)**
* **│ ├── schemas.py # Pydantic schemas**
* **│ ├── crud.py # Business logic**
* **│ └──database.py # In-memory storage**
* **│**
* **├── seed\_data.py # Initial data seeding (Contains class data)**
* **├── requirements.txt # Dependencies**
* **├── README.md # Instructions**
* **└── Documentation**

**#Note:** I divided the project into small parts which helps to implement easily and makes understandable **.**

**Code Explanation:**

**1.Data Models (models.py):**

**Fitness Class**

class FitnessClass(BaseModel):

id: int

name: str

date\_time: datetime

instructor: str

available\_slots: int

* id: Unique ID of the class.
* name: Class name
* date\_time: Scheduled time.
* instructor: Teacher name.
* available\_slots: How many people can still book.

**Booking Class**

class Booking(BaseModel):

id: int

class\_id: int

client\_name: str

client\_email: str

booked\_at: datetime

* id: Booking ID.
* class\_id: ID of the class being booked.
* client\_name and client\_email: person details
* booked\_at: Timestamp when the booking was made.

**2.API Schemas (schemas.py):**

class FitnessClassOut(BaseModel):

id: int

name: str

date\_time: datetime

instructor: str

available\_slots: int

•Used when returning class data in the API.

**Booking Request Schema**

class BookingRequest(BaseModel):

class\_id: int

client\_name: str

client\_email: EmailStr

* Used when a client wants to book a class. Ensures:
* Email is valid
* Required fields are present

**Booking Output**

class BookingOut(BaseModel):

id: int

class\_id: int

client\_name: str

client\_email: EmailStr

booked\_at: datetime

* Used when returning booking data.

**3.Database (database.py):**

fitness\_classes: List[FitnessClass] = []

bookings: List[Booking] = []

* These are Python lists used as an in-memory database. All data is stored here.

class\_id\_counter = 1

booking\_id\_counter = 1

These increment when a new class or booking is added to keep IDs unique.

**4.Logic(crud.py):**

1. List all upcoming classes:

def list\_classes() -> List[FitnessClass]:

now = datetime.now()

return [cls for cls in fitness\_classes if cls.date\_time > now]

• Filters out past classes

• Returns only upcoming ones

2. Book a Class:

def book\_class(data: BookingRequest) -> Booking:

global booking\_id\_counter

• Accepts a BookingRequest object

• Uses global to update the booking ID

selected\_class = next((cls for cls in fitness\_classes if cls.id == data.class\_id), None)

•Finds the class by class\_id.

*#Validate and Decrement Slot*

if not selected\_class:

raise ValueError("Class not found")

if selected\_class.available\_slots <= 0:

raise ValueError("No slots available")

selected\_class.available\_slots -= 1

• Throws errors if the class doesn't exist or is full.

• Reduces available slots by 1.

*#Create a Booking*

booking = Booking(

id=booking\_id\_counter,

class\_id=data.class\_id,

client\_name=data.client\_name,

client\_email=data.client\_email,

booked\_at=datetime.now()

)

bookings.append(booking)

booking\_id\_counter += 1

return booking

• Adds new booking to the list

• Increments booking ID

3. Get Bookings by Email

def get\_bookings\_by\_email(email: str) -> List[Booking]:

return [b for b in bookings if b.client\_email == email]

•Filters bookings by the user's email.

**5.FastAPI APP (main.py):**

1. Initialize App

app = FastAPI(title="Fitness Studio Booking API")

seed\_classes()

• Creates FastAPI instance

• Seeds data when the server starts

1. API Routes

*#GET /classes*

@app.get("/classes", response\_model=List[FitnessClassOut])

def get\_classes():

return list\_classes()

* Returns all upcoming classes
* Uses schema to validate response

*#POST /book*

@app.post("/book", response\_model=BookingOut)

def book(request: BookingRequest):

try:

return book\_class(request)

except ValueError as e:

raise HTTPException(status\_code=400, detail=str(e))

* Accepts booking data
* Handles errors (no class / no slots)

*#GET /bookings?email=*

@app.get("/bookings", response\_model=List[BookingOut])

def get\_bookings(email: str = Query(..., description="Client email")):

return get\_bookings\_by\_email(email)

* Takes an email as a query param
* Returns all bookings for that email

**6.Sample Data (seed\_data.py):**

Code:

def seed\_classes():

global class\_id\_counter

now = datetime.now()

classes = [

{"name": "Yoga", "instructor": "Netaji", "slots": 5},

{"name": "Zumba", "instructor": "Sai", "slots": 10},

{"name": "HIIT", "instructor": "Akash", "slots": 8},

]

* Creates 3 sample classes for testing
* Automatically assigns future times and unique IDs

Code:

for i, cls in enumerate(classes):

fitness\_classes.append(

FitnessClass(

id=class\_id\_counter,

name=cls["name"],

date\_time=now + timedelta(days=i+1),

instructor=cls["instructor"],

available\_slots=cls["slots"]

)

)

class\_id\_counter += 1

Each class is:

• Scheduled i+1 days from now

• Added to in-memory class list

**Run the App:**

**1.Install the requirements:**

pip install -r requirements.txt

**2.Run the server:**

uvicorn app.main:app --reload

**3.API Docs:**

http://127.0.0.1:8000/docs

-visit the above link for UI.