Task Assigned

FAST API , CRUD with POST,GET,PUT,DELETE APIs, Database Postgress Functions like: userslist, usercreate, update, get , delete through API end points.

FastAPI CRUD Project – Complete Notes

1. What is an API?

An **API (Application Programming Interface)** is a way for different software systems to communicate with each other.

Simple explanation:

- A client (browser, mobile app, Postman, Swagger UI) sends a request
- A server processes it and sends back a response

Example:

- Client: "Give me all users"
- Server: Returns users data in JSON

APIs usually communicate over **HTTP**.

---

2. HTTP Requests & Methods

HTTP requests define **what action** the client wants to perform.

Common HTTP Methods

Method	Purpose	Example
GET	Read data	Get all users
POST	Create data	Create a new user
PUT	Update data	Update an existing user
DELETE	Remove data	Delete a user

Example:

```
GET /users
```

Means: Fetch all users from the server.

---

3. REST API

REST (Representational State Transfer) is a **design style**, not a tool.

REST principles:

- Uses HTTP methods
- Uses URLs to represent resources
- Stateless (each request is independent)
- Uses JSON for data exchange

REST Examples from this project:

Action	Endpoint
Get all users	GET /users
Get one user	GET /users/{id}
Create user	POST /users
Update user	PUT /users/{id}
Delete user	DELETE /users/{id}

This project **follows REST rules**, so it is a **RESTful API**.

---

4. Is FastAPI a REST API?

- ❌ FastAPI is **not** a REST API
- ✅ FastAPI is a **framework used to build REST APIs**

REST = rules
FastAPI = tool to implement those rules

---

5. Why FastAPI?

FastAPI is chosen because it provides:

- High performance
- Async support
- Automatic Swagger documentation
- Built-in validation using Pydantic
- Clean and readable code

---

6. Alternatives

REST API Alternatives

- **GraphQL** – Flexible queries
- **gRPC** – High-performance internal APIs
- **SOAP** – Legacy XML-based APIs
- **WebSockets** – Real-time communication

FastAPI Alternatives

- Flask (Python)
- Django REST Framework (Python)
- Express.js (Node.js)
- Spring Boot (Java)

---

7. Project Architecture

```
project/
│
├── main.py         # API routes & logic
├── database.py     # Database setup & models
├── schemas.py      # Request & response validation
```

This separation makes the project clean and scalable.

---

8. Libraries & Frameworks Used

FastAPI

- Creates APIs
- Handles HTTP requests
- Manages dependency injection

Uvicorn

- ASGI server
- Runs the FastAPI application

SQLAlchemy

- ORM (Object Relational Mapper)
- Converts Python objects into database rows

Pydantic

- Validates API input & output data

PostgreSQL

- Relational database for persistent storage

---

9. Uvicorn & ASGI

Uvicorn

Uvicorn is an **ASGI server** that runs the FastAPI app.

Command:

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

ASGI (Asynchronous Server Gateway Interface)

- Modern interface for Python web apps
- Supports async and sync execution
- Faster and more scalable than WSGI

---

10. BaseModel vs Base

BaseModel (Pydantic)

Used for:

- Request validation
- Response formatting
- Ensuring correct data types

Example:

```
class UserCreate(BaseModel):
    name: str
    email: str
```

Base (SQLAlchemy)

Used for:

- Defining database tables
- Mapping Python classes to DB tables

Example:

```
class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True)
```

Difference

BaseModel	Base
API data validation	Database mapping
JSON ↔ Python	Python ↔ Database

---

11. Database Session

What is a Session?

A **session** is a temporary connection between the API and the database.

It:

- Executes queries
- Tracks changes
- Commits or rolls back transactions

Each API request gets its own session.

---

12. Dependency Injection (`Depends`)

```
db: Session = Depends(get_db)
```

Depends tells FastAPI to:

- Run `get_db()` first
- Inject the database session into the endpoint

This ensures:

- Clean code
- Automatic resource management

---

13. `yield db`

```
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

`yield`:

- Provides the DB session to the API
- Ensures cleanup after request completion

Used instead of `return` to safely close connections.

---

14. `.first()`

```
user = db.query(User).filter(User.id == user_id).first()
```

- Returns the first matching record
- Returns `None` if no record exists

Used for safely fetching single records.

---

15. HTTPException

```
raise HTTPException(status_code=404)
```

Used to:

- Send proper HTTP error responses
- Inform clients about errors

Common status codes:

Code	Meaning
200	OK
201	Created
404	Not Found
500	Server Error

---

16. ORM (Object Relational Mapping)

ORM allows interaction with the database using Python objects instead of SQL.

Example:

```
user = User(name="Ali", email="ali@email.com")
db.add(user)
```

---

17. APIs Created in This Project

Total APIs: **5**

HTTP Method	Endpoint	Purpose
GET	/users	List users
GET	/users/{id}	Get user
POST	/users	Create user
PUT	/users/{id}	Update user
DELETE	/users/{id}	Delete user

---

18. Sync vs Async in This Project

- APIs are **synchronous**
- Defined using `def`, not `async def`
- SQLAlchemy used is blocking

FastAPI supports async, but sync is:

- Easier
- Stable
- Commonly used

---

19. `db.commit()`

`db.commit()` permanently saves changes to the database.

Used for:

- Create
- Update
- Delete

Without commit → data is not saved.

---

20. Swagger UI

Swagger UI is auto-generated API documentation.

URL:

```
http://127.0.0.1:8000/docs
```

Features:

- Lists all APIs
- Shows request & response formats
- Allows testing APIs directly

---

21. Postman

Postman is an external API testing tool.

Difference:

Swagger	Postman
Auto-generated	Manual
Dev-friendly	QA & testing

---

22. How to Run the Project

Step 1: Install dependencies

```
pip install fastapi uvicorn sqlalchemy psycopg2-binary
```

Step 2: Run server

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

Step 3: Open Swagger

```
http://127.0.0.1:8000/docs
```

---

23. Request → Response Flow

1. Client sends HTTP request
2. FastAPI validates data
3. Dependency opens DB session
4. ORM executes query
5. `db.commit()` saves changes (if needed)
6. Response returned as JSON
7. Session closes automatically

---

24. Final Summary

This project is a **RESTful FastAPI application** implementing full CRUD operations using PostgreSQL. It uses Pydantic for data validation, SQLAlchemy as ORM, dependency injection for database management, and Uvicorn as an ASGI server. Swagger UI provides interactive API documentation, making the system clean, scalable, and production-ready.