🚀FastAPI Full Beginner-to-Project Guide

🛖Overview: The Three Learning Phases

🚃Phase 1: FastAPI Fundamentals

Goal: Understand how FastAPI works and how to create basic routes and APIs.

Key Concepts:

- What is FastAPI & why use it?
- Installing FastAPI and Uvicorn
- Basic server setup
- Creating routes
- Request/response models with Pydantic
- Swagger UI & ReDoc documentation

Phase 2: CRUD Operations with FastAPI

Goal: Learn to build real backend logic using CRUD principles and connect it with a database.

Key Concepts:

- Connecting to SQLite/PostgreSQL with SQLModel or SQLAlchemy
- Creating a database model
- CRUD (Create, Read, Update, Delete) endpoints
- Dependency injection for DB session
- Handling exceptions

🏡Phase 3: Real Project Development

Goal: Build a complete application using everything learned.

Suggested Projects:

- ✅Todo App with User Authentication
- AI SQL Chatbot Backend (for LangChain integration)
- ☢️File Upload & Download API

Phase 1: FastAPI Fundamentals

Step 1: Install FastAPI

```
pip install fastapi uvicorn
```

Step 2: Create your first app (`main.py`)

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "Hello, FastAPI!"}
```

Step 3: Run the app

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

- Visit: http://127.0.0.1:8000
- Docs: http://127.0.0.1:8000/docs (Swagger UI)
- ReDoc: http://127.0.0.1:8000/redoc

Step 4: Path Parameters

```python
@app.get("/items/{item_id}")
def read_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}
```

Step 5: Request Body with Pydantic

```python
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    description: str = None
    price: float
    tax: float = None

@app.post("/items/")
```

```
def create_item(item: Item):
    return {"item": item}
```

🚞Phase 2: CRUD Operations (SQLite Example)

Step 1: Install dependencies

```
pip install sqlmodel
```

Step 2: Define Models and DB Setup (`models.py`)

```
from sqlmodel import SQLModel, Field

class Todo(SQLModel, table=True):
    id: int | None = Field(default=None, primary_key=True)
    title: str
    description: str | None = None
    completed: bool = False
```

Step 3: Create DB and CRUD Operations (`main.py`)

```
from fastapi import FastAPI, HTTPException, Depends
from sqlmodel import Session, SQLModel, create_engine, select
from models import Todo

app = FastAPI()
database_url = "sqlite:///./test.db"
engine = create_engine(database_url, echo=True)

# Create DB tables
@app.on_event("startup")
def on_startup():
    SQLModel.metadata.create_all(engine)

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

@app.post("/todos/")
def create_todo(todo: Todo, session: Session = Depends(get_session)):
    session.add(todo)
    session.commit()
    session.refresh(todo)
    return todo

@app.get("/todos/")
def read_todos(session: Session = Depends(get_session)):
    return session.exec(select(Todo)).all()

@app.get("/todos/{todo_id}")
def read_todo(todo_id: int, session: Session = Depends(get_session)):
    todo = session.get(Todo, todo_id)
    if not todo:
        raise HTTPException(status_code=404, detail="Todo not found")
    return todo

@app.put("/todos/{todo_id}")
def update_todo(todo_id: int, updated: Todo, session: Session =
Depends(get_session)):
    db_todo = session.get(Todo, todo_id)
    if not db_todo:
        raise HTTPException(status_code=404, detail="Todo not found")
    db_todo.title = updated.title
    db_todo.description = updated.description
    db_todo.completed = updated.completed
    session.commit()
    return db_todo

@app.delete("/todos/{todo_id}")
def delete_todo(todo_id: int, session: Session = Depends(get_session)):
    todo = session.get(Todo, todo_id)
    if not todo:
        raise HTTPException(status_code=404, detail="Todo not found")
    session.delete(todo)
    session.commit()
    return {"ok": True}
```

Phase 3: Real Projects to Build

1. Todo App with Auth

- Add `users` table
- JWT authentication
- Auth middleware to protect routes

2. LangChain SQL Chatbot Backend

- Endpoint to receive natural language query
- LangChain to convert to SQL
- Execute query and return results

3. File Upload & Download API

- Use `UploadFile` and `File` from FastAPI
- Save files to local/cloud
- Generate signed download URLs

Helpful Tips

- Use `Pydantic` for all validation
- Use `Depends()` for everything reusable (DB, auth)
- Swagger UI is your friend during dev
- Uvicorn + `--reload` helps during iteration
- Use `SQLModel` for simple apps or `SQLAlchemy` for more control

Final Thoughts

FastAPI is clean, async-friendly, and production-grade. Once you've done this guide:

- You're ready to build APIs at scale
- Integrate with AI, ML, databases
- Deploy on Render, Vercel, or Docker

Now go build something epic!