**Developing a Fully Functional API with FastAPI for a Personal Portfolio Website**

In this guide, we will create a fully functional API using FastAPI in Python for a personal portfolio website. The API will include endpoints for managing projects, blog posts, and contact information. We will use SQLite as our database to store the data.

### **Step 1: Setting Up the Environment**

First, ensure you have Python installed on your machine. You can check this by running:

python --version

Next, create a new directory for your project and navigate into it:

mkdir portfolio\_api

cd portfolio\_api

Now, create a virtual environment and activate it:

python -m venv venv

source venv/bin/activate # On Windows use `venv\Scripts\activate`

Install the required packages:

pip install fastapi uvicorn sqlalchemy sqlite3 pydantic

### **Step 2: Creating the Database Models**

Create a file named models.py to define our database models using SQLAlchemy.

# models.py

from sqlalchemy import Column, Integer, String, Text, create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

Base = declarative\_base()

class Project(Base):

\_\_tablename\_\_ = 'projects'

id = Column(Integer, primary\_key=True, index=True)

title = Column(String(100), index=True)

description = Column(Text)

url = Column(String(200))

class BlogPost(Base):

\_\_tablename\_\_ = 'blog\_posts'

id = Column(Integer, primary\_key=True, index=True)

title = Column(String(100), index=True)

content = Column(Text)

class ContactInfo(Base):

\_\_tablename\_\_ = 'contact\_info'

id = Column(Integer, primary\_key=True, index=True)

email = Column(String(100))

phone\_number = Column(String(15))

# Database setup

DATABASE\_URL = "sqlite:///./portfolio.db"

engine = create\_engine(DATABASE\_URL)

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

def init\_db():

Base.metadata.create\_all(bind=engine)

### **Step 3: Creating the FastAPI Application**

Create a file named main.py where we will set up our FastAPI application and define the endpoints.

# main.py

from fastapi import FastAPI, Depends, HTTPException

from sqlalchemy.orm import Session

from models import init\_db, SessionLocal, Project as ProjectModel, BlogPost as BlogPostModel, ContactInfo as ContactInfoModel

app = FastAPI()

# Initialize the database

init\_db()

# Dependency to get DB session

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

@app.post("/projects/")

def add\_project(project: ProjectModel, db: Session = Depends(get\_db)):

db.add(project)

db.commit()

db.refresh(project)

return project

@app.get("/projects/")

def get\_projects(db: Session = Depends(get\_db)):

return db.query(ProjectModel).all()

@app.get("/projects/{project\_id}")

def get\_project(project\_id: int, db: Session = Depends(get\_db)):

project = db.query(ProjectModel).filter(ProjectModel.id == project\_id).first()

if not project:

raise HTTPException(status\_code=404, detail="Project not found")

return project

@app.put("/projects/{project\_id}")

def edit\_project(project\_id: int, updated\_project: ProjectModel, db: Session = Depends(get\_db)):

project\_query = db.query(ProjectModel).filter(ProjectModel.id == project\_id)

if not project\_query.first():

raise HTTPException(status\_code=404, detail="Project not found")

project\_query.update(updated\_project.dict())

db.commit()

return project\_query.first()

@app.delete("/projects/{project\_id}")

def delete\_project(project\_id: int, db: Session = Depends(get\_db)):

project\_query = db.query(ProjectModel).filter(ProjectModel.id == project\_id)

if not project\_query.first():

raise HTTPException(status\_code=404, detail="Project not found")

project\_query.delete()

db.commit()

return {"detail": "Project deleted"}

# Blog Posts Endpoints (similar structure as Projects)

@app.post("/blog\_posts/")

def add\_blog\_post(blog\_post: BlogPostModel, db: Session = Depends(get\_db)):

# Similar implementation as add\_project

@app.get("/blog\_posts/")

def get\_blog\_posts(db: Session = Depends(get\_db)):

# Similar implementation as get\_projects

@app.get("/blog\_posts/{post\_id}")

def get\_blog\_post(post\_id: int):

# Similar implementation as get\_project

@app.put("/blog\_posts/{post\_id}")

def edit\_blog\_post(post\_id:int):

# Similar implementation as edit\_project

@app.delete("/blog\_posts/{post\_id}")

def delete\_blog\_post(post\_id:int):

# Similar implementation as delete\_project

# Contact Information Endpoints (similar structure)

@app.post("/contact\_info/")

def add\_contact\_info(contact\_info: ContactInfoModel):

# Implementation here

@app.put("/contact\_info/{info\_id}")

def edit\_contact\_info(info\_id:int):

# Implementation here

@app.delete("/contact\_info/{info\_id}")

def delete\_contact\_info(info\_id:int):

# Implementation here

if \_\_name\_\_ == "\_\_main\_\_":

import uvicorn

uvicorn.run(app)

### **Step 4: Running the Application**

To run your application locally:

uvicorn main:app --reload

You can now access your API at http://127.0.0.1:8000/docs to see the automatically generated documentation provided by FastAPI.

### **Step 5: Pushing Code to GitHub**

1. Initialize a git repository in your directory:  
   git init

Add your files and commit them:  
git add .

1. git commit -m "Initial commit of portfolio API"
2. Create a new repository on GitHub and follow their instructions to push your local repository to GitHub.

### **Conclusion**

You now have a fully functional API built with FastAPI that manages projects and blog posts along with contact information for your personal portfolio website. This setup allows you to easily expand or modify features in the future.