

BACKEND DEVELOPMENT

(FASTAPI + POSTGRES + RENDER)

1. Project Overview

This backend project powers the frontend portfolio application by providing a **fully functional, secure, and extensible API**. It is built using **FastAPI**, one of the fastest Python frameworks, combined with **SQLAlchemy**, **PostgreSQL**, and **Render Cloud Deployment**.

“Building a Secure, Scalable & Production-Ready API Layer”

The backend manages:

- CRUD operations for project data
- File uploads (PDFs, images)
- Database interactions
- Validation & error handling
- Production-ready deployment

Its goal is to deliver a **robust, modular, and secure backend foundation** for any full-stack application.

2. Tech Stack Used

Backend Framework

FastAPI - Chosen for its speed, async support, automatic documentation, and clean architecture.

Database

PostgreSQL (Cloud DB on Render) - Reliable, scalable, ideal for structured data and production apps.

ORM

SQLAlchemy - Simplifies database models, relationships, and queries.

Cloud Deployment

Render - Used for hosting both the API and the database and it includes auto-deploy, SSL, logs, health checks.

File Handling

UploadFile / File (FastAPI) - For accepting and storing documents.

CORS Middleware

Enables secure communication between frontend and backend.

Pydantic Models

Used for type safety, validation, and serialization.

3. Folder Structure

```
/project-backend
    ├── main.py
    ├── models.py
    ├── schemas.py
    ├── database.py
    ├── crud.py
    ├── requirements.txt
    ├── docs/
    ├── uploads/
    └── routers/
        └── projects.py
```

4. API Features

4.1 CRUD Operations for Projects

Backend supports all CRUD operations:

- Create Project

- Accepts:
- title
- description
- tech stack
- GitHub link
- Demo link
- Project document (PDF / image)

- Fetch All Projects

- Used in your frontend portfolio carousel.

- Fetch Project by ID

- Used in /project/:id page.

- Update Project

- Allows flexible editing through PATCH.

- Delete Project

- Your delete button triggers:

4.2 File Upload Handling

This includes:

- Uploading project documents via:

```
@app.post("/demo-projects/upload")
```

- Saving files to /uploads
- Serving files publicly using:

```
app.mount("/docs", StaticFiles(directory="docs"), name="docs")
```

4.3 Database Operations

Using SQLAlchemy ORM:

- Model definitions
- Mapping to PostgreSQL tables
- Query execution
- Auto-commit transactions
- Model relationships (future expandable)

5. Why These Technologies Are Used

FastAPI

- Extremely fast (async support)
- Auto-generated swagger documentation
- Clean, modern architecture
- Easy testing and scaling
- Perfect match for TypeScript frontend

SQLAlchemy

- Eliminates raw SQL complexity
- Prevents SQL injection
- Makes migrations easier
- Helps maintain clean model structure

PostgreSQL

- Industry-standard
- Supports large datasets
- Perfect for relational apps
- Reliable for production environments

Render Cloud

- Handles deployment complexity
- Auto SSL
- Health monitoring
- Restart-on-failure
- Easy CI/CD integration
- Free tier available (budget friendly)

6. Key Backend Responsibilities

Data Persistence

Safely stores project info + documents.

Data Validation

Ensures correct input before saving.

API Security

Prevents origin mis-match using CORS config.

Error Handling

422, 404, 500 handlers for stability.

Scalability

Modular architecture allows:

- Adding users
- Adding authentication
- Adding project categories

- Adding blog system

7. Challenges Solved

File upload 422 error

Solved by:

- Using Form(...) for text fields
- Using UploadFile = File(...) for documents
- Sending proper multipart/form-data from frontend.

Avoiding page break images

Handled by resizing using proper JSON parsing.

Rendering images in carousel

Fixed by parsing image_url as array or single string.