# 📝 Software Engineering Blueprint for Python FastAPI Projects

This document outlines best practices, coding guidelines, and security recommendations for maintaining high-quality FastAPI-based applications.

## 1️⃣ General Coding Standards

Follow PEP8: Ensure code follows PEP8 style guidelines.

Use Type Hinting: All function parameters and return types must be type-hinted.

Consistent Naming: Use snake\_case for variables and functions, and PascalCase for class names.

Example:

def get\_user(user\_id: int) -> dict:  
 pass

Use Dependency Injection for Database Sessions

Use HTTP Status Codes Properly

Error Handling is Mandatory

Example:

@router.get("/users/{user\_id}")  
def get\_user(user\_id: int, db: Session = Depends(get\_db)):  
 try:  
 user = db.query(User).filter(User.id == user\_id).first()  
 if not user:  
 raise HTTPException(status\_code=404, detail="User not found")  
 return user  
 except Exception:  
 raise HTTPException(status\_code=500, detail="Internal Server Error")

Use SQLAlchemy ORM for Queries

Optimize Queries

Prevent SQL Injection

Example of Bad Practice:

users = db.query(User).all()  
for user in users:  
 user.is\_active = True  
 db.commit()

Example of Good Practice:

db.query(User).update({'is\_active': True})  
db.commit()

Use OAuth2 for Authentication

Validate Input Data

Limit API Rate Usage

Example:

@router.post("/login")  
def login(user\_data: LoginSchema):  
 if user\_data.password != "expected\_password":  
 raise HTTPException(status\_code=401, detail="Unauthorized")

Use Logging Instead of print()

Monitor API Performance

Example:

import logging  
  
logging.basicConfig(level=logging.INFO)  
logger = logging.getLogger(\_\_name\_\_)  
  
logger.info("User login attempt detected")

## 📌 Summary

|  |  |
| --- | --- |
| Section | Best Practice |
| General | Follow PEP8, use type hints |
| API Design | Use proper status codes, dependency injection |
| Database | Use SQLAlchemy ORM, optimize queries |
| Security | Use OAuth2, validate input data |
| Logging | Use structured logs instead of print() |