📘 System Design with Python

Part 15: Security Best Practices in System Design

Why Security Matters?

🔓 One vulnerability can expose entire systems
⚡ Attacks scale faster than apps
🏦 Security breaches cost $$$ & user trust

✅ A secure system is not optional — it’s foundational.

Real-World Analogy

💡 Think of a Newspaper Agency

- Lock (Authentication)
- Guard (Authorization)
- CCTV (Monitoring)
- Insurance (Backups)

Your software system needs the same protections.

Common Web Threats

- 💉 SQL Injection → Attacker manipulates DB queries
- 🖼️ XSS (Cross-Site Scripting) → Injects malicious JS
- 🪝 CSRF (Cross-Site Request Forgery) → Tricks user into unintended actions
- 🌍 CORS Misconfigurations → Unauthorized domain access

Secure Authentication

FastAPI + JWT Example

```
from fastapi import FastAPI, Depends
from fastapi.security import OAuth2PasswordBearer
from jose import JWTError, jwt
import time

app = FastAPI()
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")
SECRET_KEY = "mysecret"

def create_token(data: dict):
    to_encode = data.copy()
    to_encode.update({"exp": time.time() + 3600})
    return jwt.encode(to_encode, SECRET_KEY, algorithm="HS256")

@app.get("/secure-data/")
async def secure_data(token: str = Depends(oauth2_scheme)):
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=["HS256"])
        return {"msg": "Access granted", "user": payload}
    except JWTError:
        return {"error": "Invalid or expired token"}
```

Always use hashed passwords (bcrypt)

Never store plain tokens

SQL Injection Defense

Vulnerable:

```
"SELECT * FROM users WHERE name = '" + user_input + "'"
```

Safe with parameterized queries:

```
cursor.execute("SELECT * FROM users WHERE name = %s", (user_input,))
```

XSS Defense

Sanitize user inputs (bleach library in Python)

Use HTML escaping in templates

Example:

```
from markupsafe import escape
user_input = "<script>alert(1)</script>"
print(escape(user_input))  # &lt;script&gt;alert(1)&lt;/script&gt;
```

CSRF Defense

- Generate unique CSRF tokens for forms
- Validate on every request

Example with FastAPI middleware:

```
@app.middleware("http")
async def csrf_protect(request, call_next):
    if request.method in ["POST", "PUT", "DELETE"]:
        token = request.headers.get("X-CSRF-Token")
        if not token or token ≠ "expected_token":
            return JSONResponse(status_code=403, content={"error": "CSRF blocked"})
    return await call_next(request)
```

CORS Protection

```
from fastapi.middleware.cors import CORSMiddleware

app.add_middleware(
    CORSMiddleware,
    allow_origins=["https://myfrontend.com"],
    allow_methods=["GET", "POST"],
    allow_headers=["Authorization"],
)
```

Never allow * in production

HTTPS & TLS

- Always encrypt traffic (Let's Encrypt free TLS certs)
- Redirect all HTTP → HTTPS
- Secure cookies: HttpOnly, Secure, SameSite

Secrets Management

- ❌ Don't hardcode secrets in code
- ✅ Use environment variables / secret managers

Example with python-dotenv:

```
from dotenv import load_dotenv
import os

load_dotenv()
db_password = os.getenv("DB_PASSWORD")
```

Extra Best Practices

- 🔒 Rate limiting (protect from brute force)
- 🧑‍💻 Regular security audits
- 🛡️ Dependency scanning (pip-audit)
- 🗂️ Database encryption (sensitive data at rest)

Summary

- Secure APIs with JWT/OAuth2
- Prevent SQL Injection, XSS, CSRF, CORS attacks
- Enforce HTTPS & secrets management
- Security = continuous process, not one-time setup

What’s Next?

Part 16: Designing a Scalable System from Scratch

📁 Case Study (e.g., “Design YouTube” or “Design Blog App”)

- HLD + LLD breakdown
- Components, APIs, DB, caching, security
- End-to-end integration