

Authentication & Authorisation in a Software



Secure, Scalable, and Professional

A comprehensive guide to implementing robust security in financial applications



jwt-fastapi.zip

Agenda

Core Concepts

Authentication vs Authorization

JWT Deep Dive

Why JSON Web Tokens?

Java Implementation

Spring Security & JWT

Python Implementation

FastAPI & JWT

Best Practices

Security & Pro Tips

Conclusion

Key Takeaways

Authentication vs Authorization

Authentication	Authorization
Confirms identity (who someone is), usually via credentials like passwords or biometrics.	Determines permissions (what resources or actions are allowed) after authentication
Showing an ID to enter a building	Which all sections of building you can access
Best Practice – Authenticate first as this is a low cost operation	

Evolution

Server User Accounts: Initial systems relied on server OS user credentials.

Credentials in Application DB: Apps stored username-passwords directly in their databases.

SSO & Identity Providers: Moved to central authentication (Single Sign-On) with providers like Okta, Auth0, or Active Directory, using SAML, OAuth, or OIDC protocols.

Token-Based (JWT, OIDC): Tokens like JWT allow stateless session handling and cross-site authorization.

Modern Context: Now, federated logins, device-based credentials, and strong authentication are common

Password vs. Passkey

Password		Passkey
Nature	User-created secret string	Device-bound cryptographic key pair
Storage	Showing an ID to enter a building	Stored on server (hashed or encrypted)
Security	Prone to phishing, brute-force, leaks	Resistant to phishing, device-bound, uses biometrics
User action	Must remember and type	No need to remember; can use face/fingerprint/etc.
Adoption	Universal	Growing, but not yet universal

Alternatives Gaining Ground

Opaque Reference Tokens: These random strings (used with OAuth2) require server validation for each request, combining much of the session model's revocability and centralized control with the flexibility needed for APIs.

PASETO: This new token standard is designed to be "secure by default" and addresses many JWT weaknesses, potentially positioning itself as a more resilient alternative for stateless authentication in "next gen" systems

Session Cookies: These are also gaining grounds with enhanced security added. This is due to their simplicity and inherent scalability.

Why JSON Web Tokens (JWT)?

Digital Passport for Your web App 🛂

JWT Structure: `header.payload.signature`

Header

Metadata about the token (algorithm, type)

Payload

Claims about the user (ID, role, expiration)

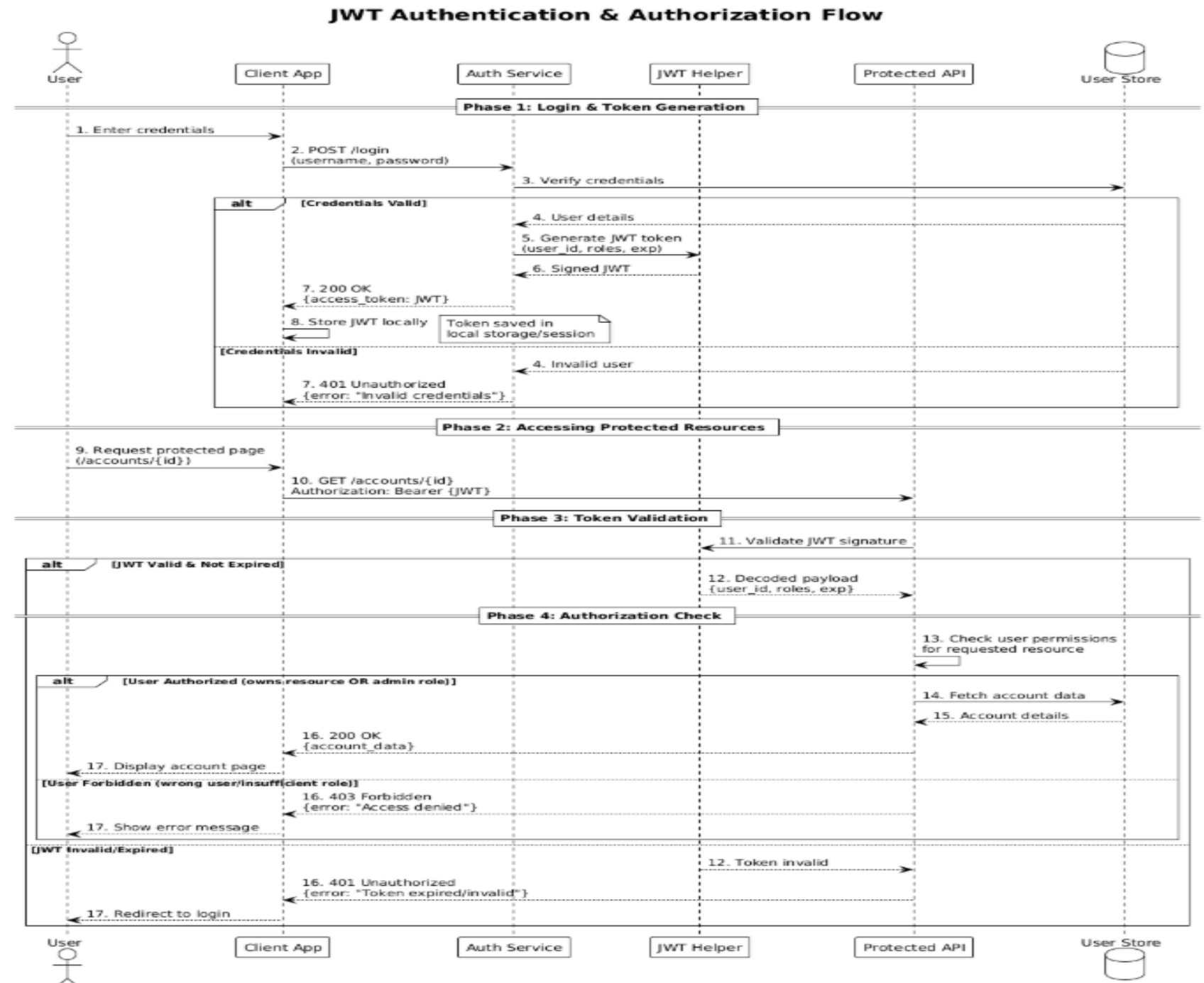
Signature

Cryptographic verification of authenticity

Why is it Perfect for web APIs?

- **Stateless:** No server-side session storage needed
- **Self-Contained:** User info and permissions in the token
- **Secure:** Cryptographic signature ensures integrity
- **Scalable:** Excellent for distributed systems

JWT Lifecycle: Creation + Secured Endpoint Access



Exercise 1: Spot the Security Flaw

A developer adds an account number to JWT payload:

```
{ "sub": "jane.doe", "account_number": "1234567890123456", "role": "CUSTOMER", "exp": 1672531199 }
```



Question: What's the primary security risk?



Answer:

JWT payload is Base64Url encoded, NOT encrypted! Anyone with the token can decode and read the account number. Never store sensitive data in JWT payload - only non-sensitive claims like user ID and roles.

Authentication Schemes: Choose the Right Tool

JWT/OAuth2 (Recommended for APIs)

How: User logs in → gets JWT → presents in Authorization header

Perfect for: Mobile and web banking frontends

API Keys

How: Unique string sent with every request

Perfect for: Server-to-server communication

Session Cookies

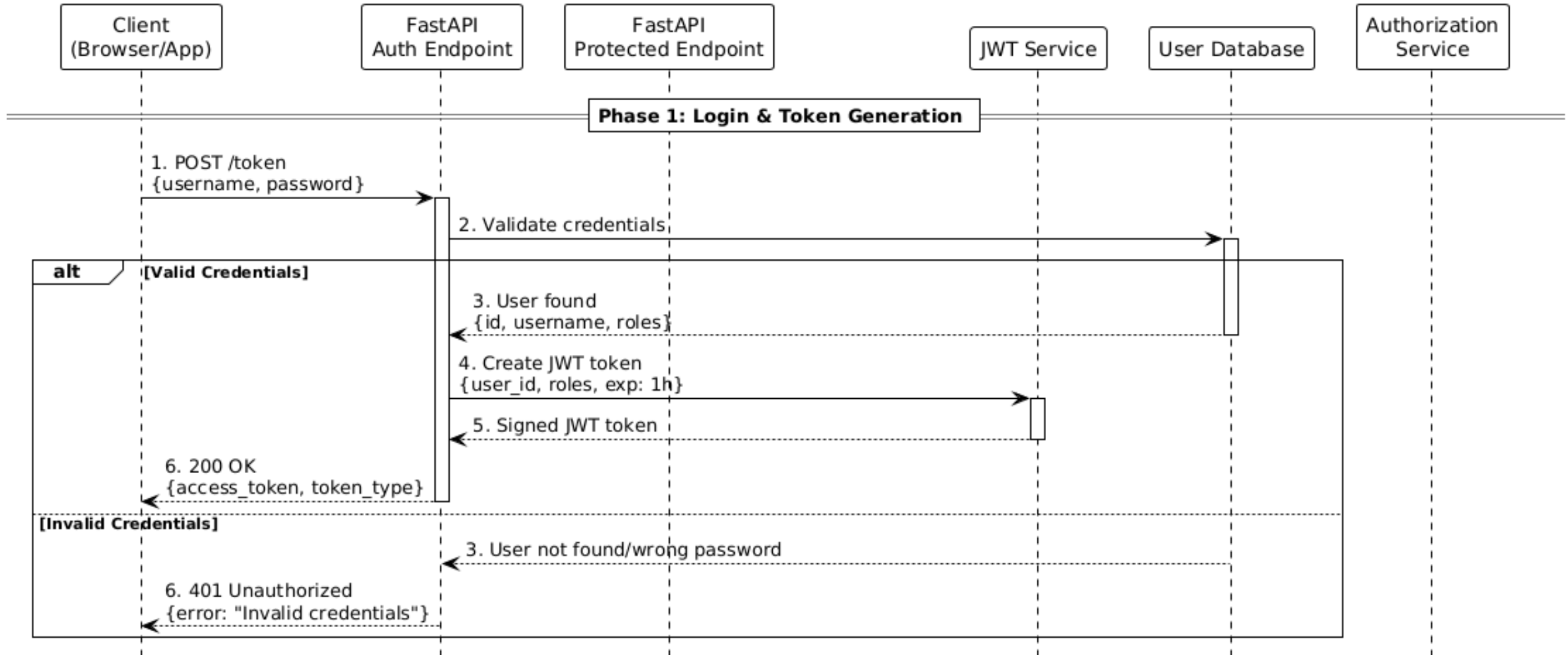
How: Server stores session, sends ID as cookie

Perfect for: Traditional server-rendered web apps

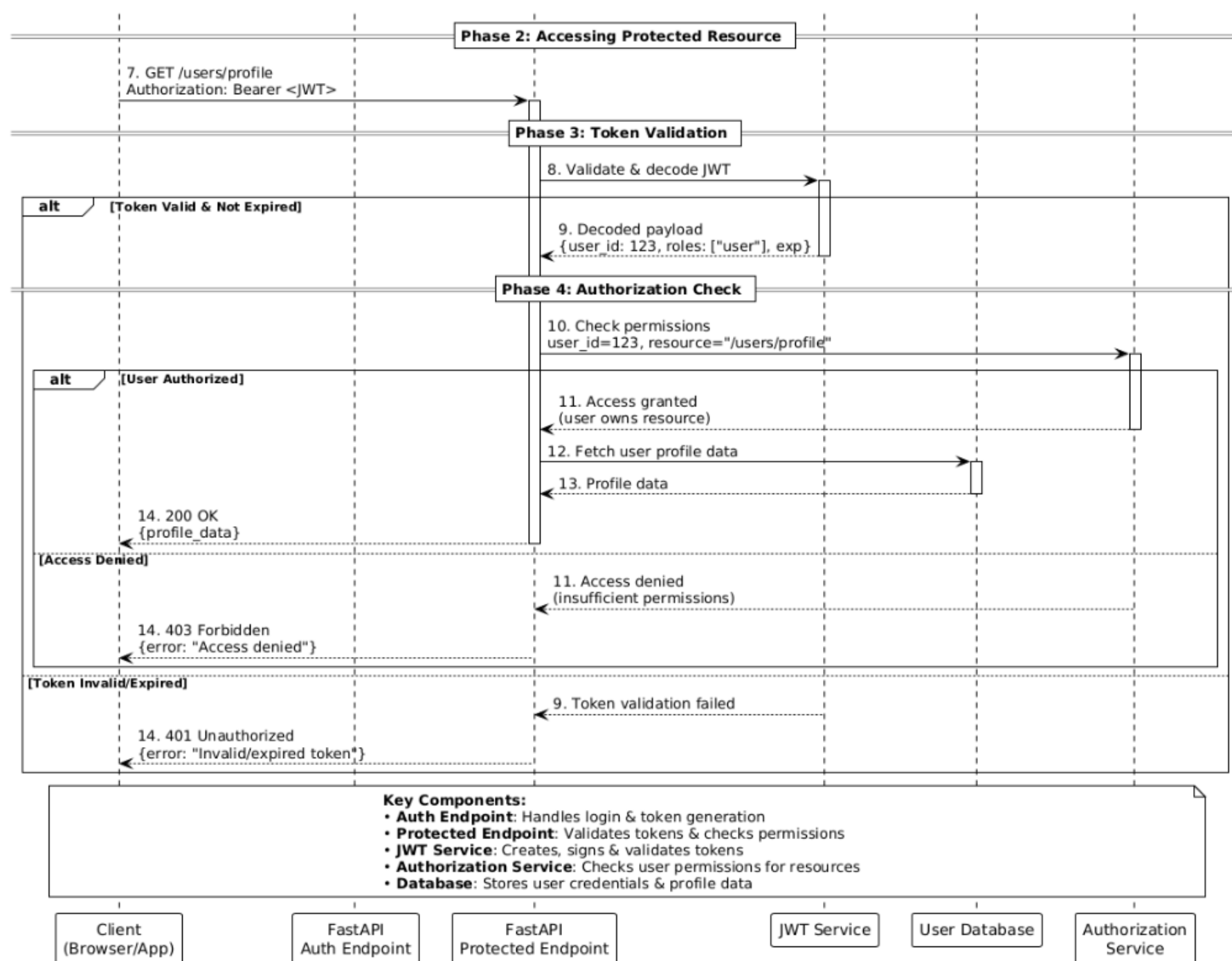
JWT Authentication & Authorisation Flow

FastAPI Implementation - Security Architecture Training

FastAPI JWT Authentication & Authorization Flow



Contd...



JWT Authentication & Authorization Flow

Phase 1: Login & Token Generation

- Client submits credentials to /token endpoint
- FastAPI validates credentials against database
- JWT Service creates signed token with user info & roles
- Token returned to client (expires in 1 hour)

Phase 2: Accessing Protected Resources

- Client includes JWT in Authorization header
- Protected endpoint receives Bearer token
- Request forwarded to token validation service

Phase 3: Token Validation

- JWT Service validates signature & expiration
- Decodes payload: user_id, roles, permissions
- Returns validation result to protected endpoint

Phase 4: Authorization Check

- Authorization Service checks user permissions
- Verifies resource access rights (owns resource?)
- Returns 200 OK with data OR 403 Forbidden

Key Components

Auth Endpoint

Handles login & token generation

Protected Endpoint

Validates tokens & serves resources

JWT Service

Creates, signs & validates tokens

Authorization Service

Checks user permissions for resources

User Database

Stores credentials & profile data

Security

Tokens are signed and expire after 1 hour. No credentials stored on client.

Error Handling

Clear HTTP status codes: 401 (unauthorized), 403 (forbidden), 200 (success).

Scalability

Stateless tokens enable horizontal scaling without session storage.

Best Practices

Role-based access control with proper separation of concerns.

Python Code Examples

Creating JWT Token

```
def create_access_token(data: dict) -> str:
    to_encode = data.copy()
    to_encode.update({"exp": datetime.now(timezone.utc) + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
```

Securing Endpoints with Dependencies

```
router = APIRouter(prefix="/accounts", tags=["accounts"])
```

[CodeMate](#) | [Qodo Gen](#): Options | [Test this function](#)

```
@router.get("/{account_id}")
```

```
async def get_account(account_id: str, current=Depends(role_required("customer"))):
    return {"account_id": account_id, "owner": current["username"], "role": current["role"]}
```

[CodeMate](#) | [Qodo Gen](#): Options | [Test this function](#)

```
@router.get("/admin/{account_id}")
```

```
async def admin_get_account(account_id: str, current=Depends(role_required("admin"))):
    return {"account_id": account_id, "requested_by": current["username"], "role": current["role"]}
```

You, 6 hours ago • adding project for JWT and Fast API

Best Practices & Pro-Level Tips

Strong Secret Keys

Long, complex keys stored securely in environment variables

Short Expiration Times

15-60 minutes for access tokens minimizes theft damage

Refresh Token Flow

Long-lived refresh tokens for seamless user experience

Algorithm Specification

Always specify expected algorithm (HS256, RS256)

Secure Storage

HttpOnly cookies for web, secure storage for mobile

No Sensitive Data

NEVER put PII or financial data in JWT payload

Key Takeaways

AuthN vs AuthZ

Identity verification vs Permission checking - you need both!

Security First

Never expose sensitive data in JWT payload

JWT is the Standard

Stateless, secure, self-contained solution for modern APIs

Use Frameworks

Spring Security & FastAPI provide battle-tested tools

Master the Flow

Login → Token → Validation → Authorization

Layered Security

JWT + secure config + validation = robust system

Build Secure, Scale Confidently! 

Mini Project - Authentication & Authorization in Action



MiniProject_Authentication_Java.zip



Building a Secure Personal Banking API



jwt-fastapi.zip

Audience

Java & Python Developers

Goal

Hands-on JWT Implementation

Duration

45 Minutes

The Core Concepts: A Quick Refresher

Authentication (AuthN)

Who are you?

The process of verifying a user's identity.

Analogy: Showing your ID card to a security guard to enter a building.

In our App: A user provides username and password to prove their identity.

Authorization (AuthZ)

What are you allowed to do?

The process of verifying if a user has permission to access a specific resource.

Analogy: Your keycard only grants access to specific floors or rooms.

In our App: A logged-in user can view their own account but not others'.

Hands-On Lab: Secure API

Your Mission

Build a secure REST API for a mini personal banking system. Implement a login endpoint that issues a JWT and use that token to protect other endpoints.

Core Features to Implement:

- ✓ **User Model:** Simple user with username, password, and role
- ✓ **Login Endpoint:** Authenticates user and returns JWT
- ✓ **Protected Endpoints:** Secure endpoints accessible only with a valid JWT

This exercise covers the entire Authentication and Authorisation flow!

The Auth Flow in Our App

1

Client Login

User sends username & password to POST /api/login

2

Server Validates & Issues JWT

API verifies credentials and generates a signed JWT containing user claims (like role)

3

Client Stores JWT

Client application receives and stores the JWT securely

The Auth Flow in Our App

4

Client Requests Protected Resource

Client makes a call to GET /api/account/details, including JWT in Authorization header

5

Server Validates JWT & Authorizes

API middleware validates JWT signature, expiration, and checks user role permissions. If valid, processes request; otherwise returns error.

Tech Stack & Key Libraries

Java Developers (Spring Boot)

Framework & Dependencies:

- ✓ Spring Boot with Spring Security
- ✓ spring-boot-starter-security
- ✓ io.jsonwebtoken:jjwt-api

Key Concepts:

- ✓ **SecurityFilterChain:** Configure public/protected endpoints
- ✓ **JwtRequestFilter:** Custom filter for JWT validation
- ✓ **UserDetailsService:** Load user data for authentication

Python Developers (FastAPI)

Framework & Dependencies:

- ✓ FastAPI
- ✓ uvicorn, python-jose, passlib
- ✓ python-jose for JWT handling

Key Concepts:

- ✓ **OAuth2PasswordBearer:** Extract token from header
- ✓ **Dependency Injection:** get_current_user function
- ✓ **Route Decorators:** Protect routes with dependencies


Conclusion & Key Learnings

What We Accomplished

- ✓ Built a functional secure API for banking
- ✓ Implemented complete Auth flow
- ✓ Hands-on JWT implementation
- ✓ Protected sensitive endpoints

Key Learnings

- ✓ **AuthN vs AuthZ:** Login vs Permission checking
- ✓ **JWTs:** Stateless, verifiable user information
- ✓ **Framework Power:** Spring Security & FastAPI
- ✓ **Security Best Practices:** Token validation & role-based access

 Congratulations!

You now have the fundamental skills to secure any modern web API!