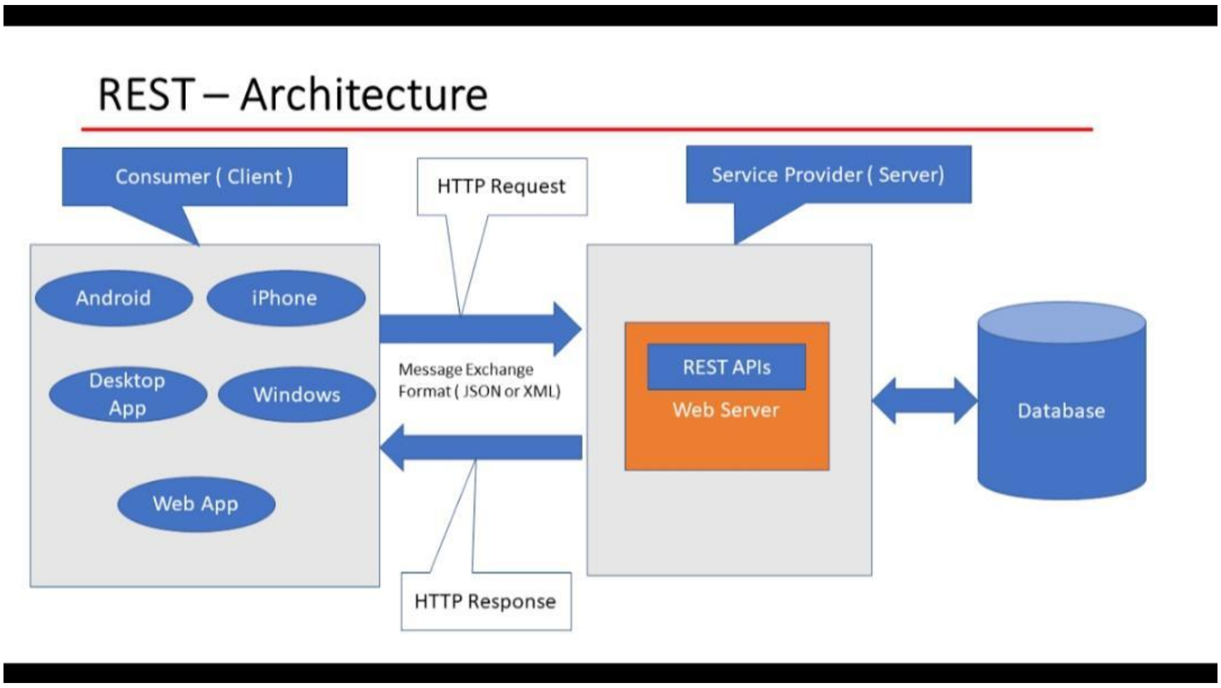
Shashank Vishwakarma

Below are clear, professional architecture diagrams

Diagram 1: Secure HTTP API Architecture (Basic)

Architecture Flow

```
Client (Browser / Mobile App)
        |
     HTTPS
        |
API Server (Spring Boot / Node)
        |
Authentication Layer (JWT / OAuth2)
        |
Business Logic
        |
Database
```

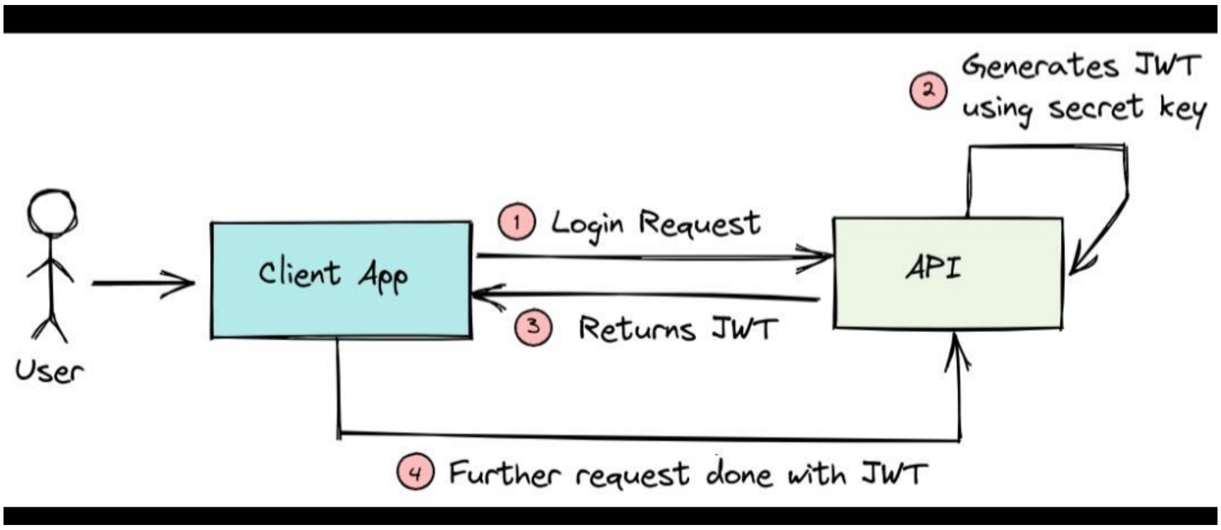


Security Highlights

- HTTPS encrypts traffic
- Authentication before business logic
- Authorization checks per request
- No direct DB exposure

All API requests must pass through transport security, authentication, and authorization layers before accessing data.

Diagram 2: JWT-Based Authentication Flow


② Generates JWT using secret key
User
Client App
① Login Request
API
③ Returns JWT
④ Further request done with JWT

Step-by-Step Flow

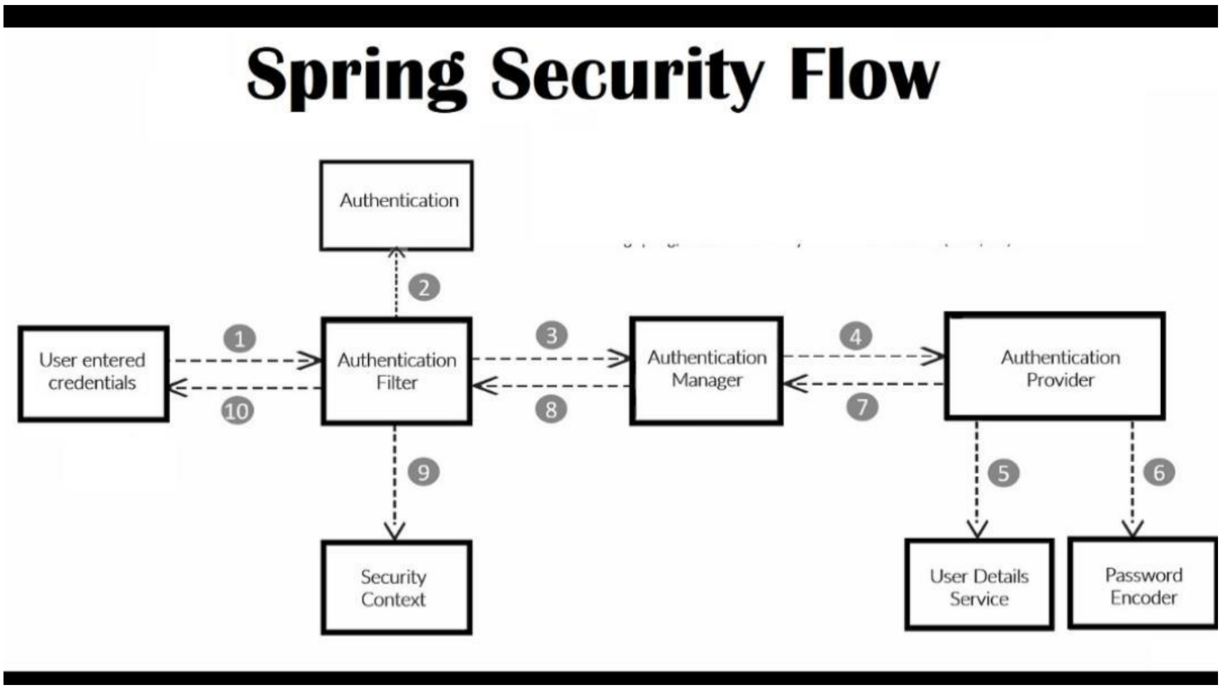
```
1. User → Login API (username/password)
2. Auth Server → Validate credentials
3. Auth Server → Issue JWT
4. Client → API Request + JWT
5. API → Validate JWT signature & expiry
6. API → Authorize → Return data
```

Security Highlights

- Stateless authentication
- Token expiry limits damage
- No session storage on server

JWT enables scalable, stateless authentication suitable for modern microservices.

Diagram 3: Spring Security Request Flow (Internal)



Internal Flow

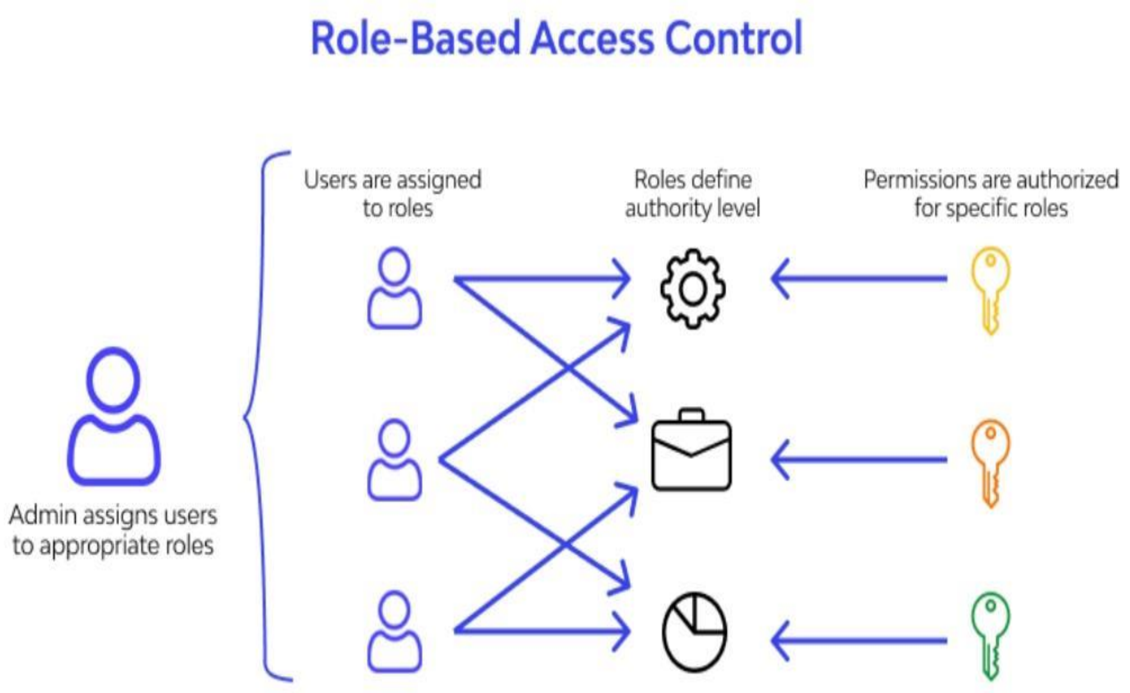
```
HTTP Request
    ↓
SecurityFilterChain
    ↓
JWT Authentication Filter
    ↓
AuthenticationManager
    ↓
UserDetailsService
    ↓
SecurityContext
    ↓
Controller
```

Security Highlights

- Filters run **before controllers**
- SecurityContext is thread-local
- Unauthorized requests stop early

Spring Security enforces authentication and authorization before any controller logic executes.

Diagram 4: Role-Based Access Control (RBAC)


Role-Based Access Control
Users are assigned to roles
Roles define authority level
Permissions are authorized for specific roles
Admin assigns users to appropriate roles

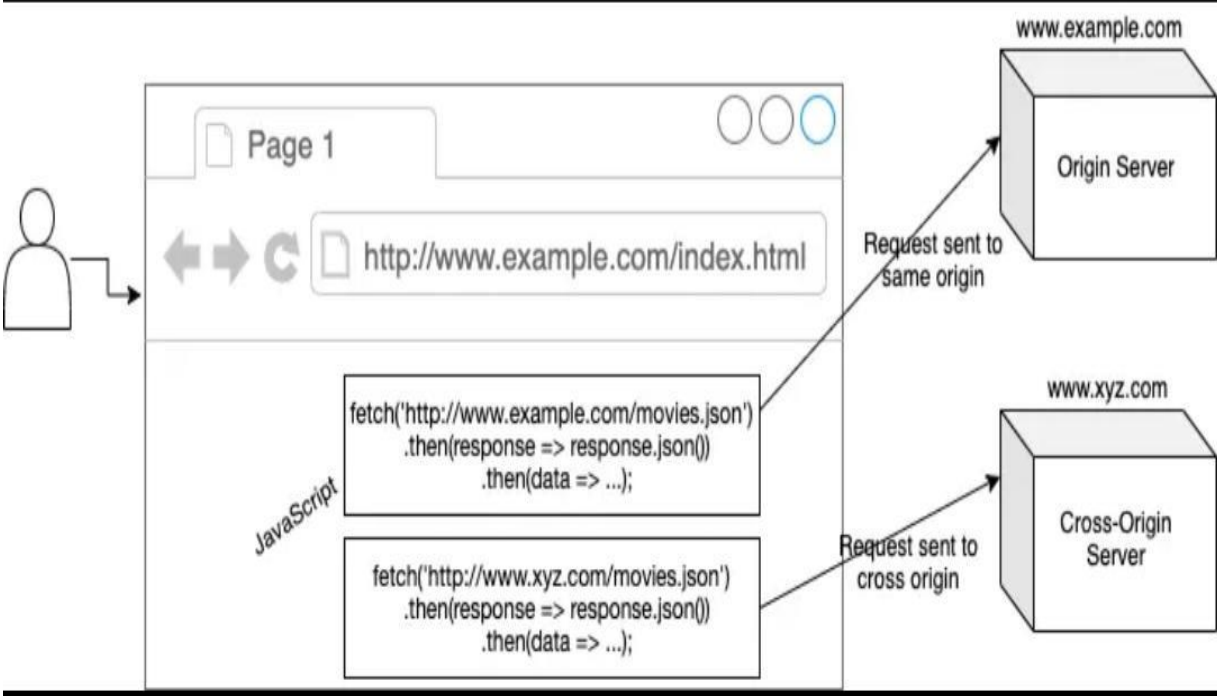
Flow

```
User → JWT (Roles embedded)
        ↓
API Authorization Layer
        ↓
Role Check
  ├── ADMIN → Full access
  ├── USER → Limited access
  └── GUEST → Public APIs
```

Security Highlights

- Fine-grained access control
- Role checks at API & method level
- Prevents privilege escalation

Diagram 5: CORS & Browser Security Flow


Page 1
http://www.example.com/index.html
JavaScript
fetch('http://www.example.com/movies.json')
.then(response => response.json())
.then(data => ...);
fetch('http://www.xyz.com/movies.json')
.then(response => response.json())
.then(data => ...);
www.example.com
Origin Server
Request sent to same origin
www.xyz.com
Cross-Origin Server
Request sent to cross origin

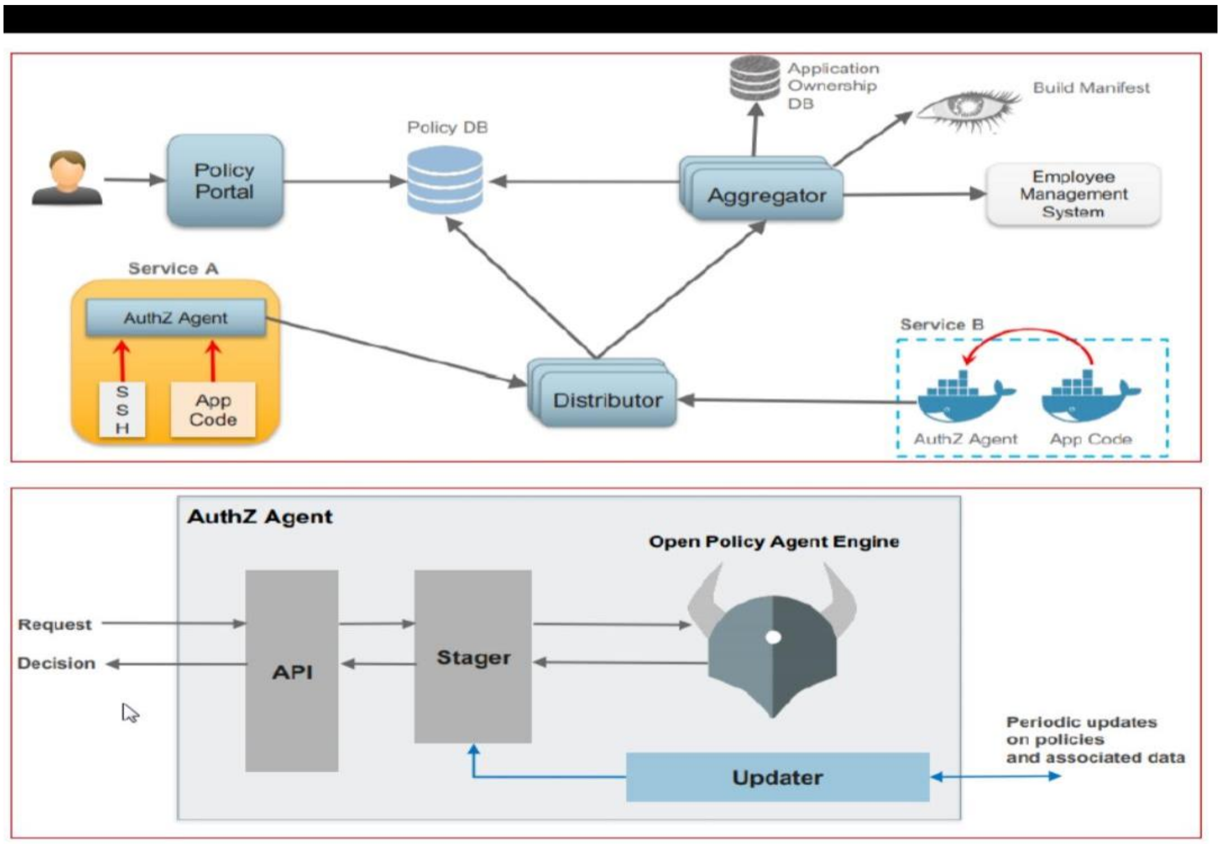
Browser Flow

```
Browser
  ↓ OPTIONS (Preflight)
API Server
  ↓ CORS Headers
Browser
  ↓ Actual Request (GET/POST)
API Server
```

Security Highlights

- Blocks unauthorized origins
- Protects authenticated browser sessions
- Only trusted domains allowed

Diagram 6: Microservices Security with API Gateway



Architecture

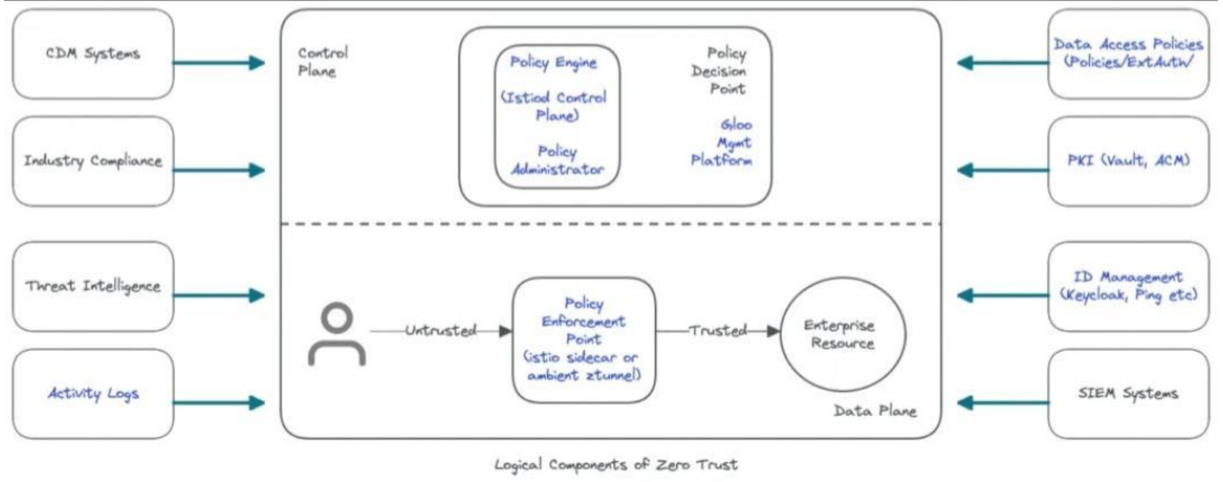
```
Client
  ↓
API Gateway
  ├── Authentication
  ├── Rate Limiting
  ├── Logging
  ↓
Microservices
  ├── User Service
  ├── Order Service
  ├── Payment Service
```

Security Highlights

- Single entry point
- Centralized security

- Reduced attack surface

Diagram 7: Zero Trust Microservices Architecture



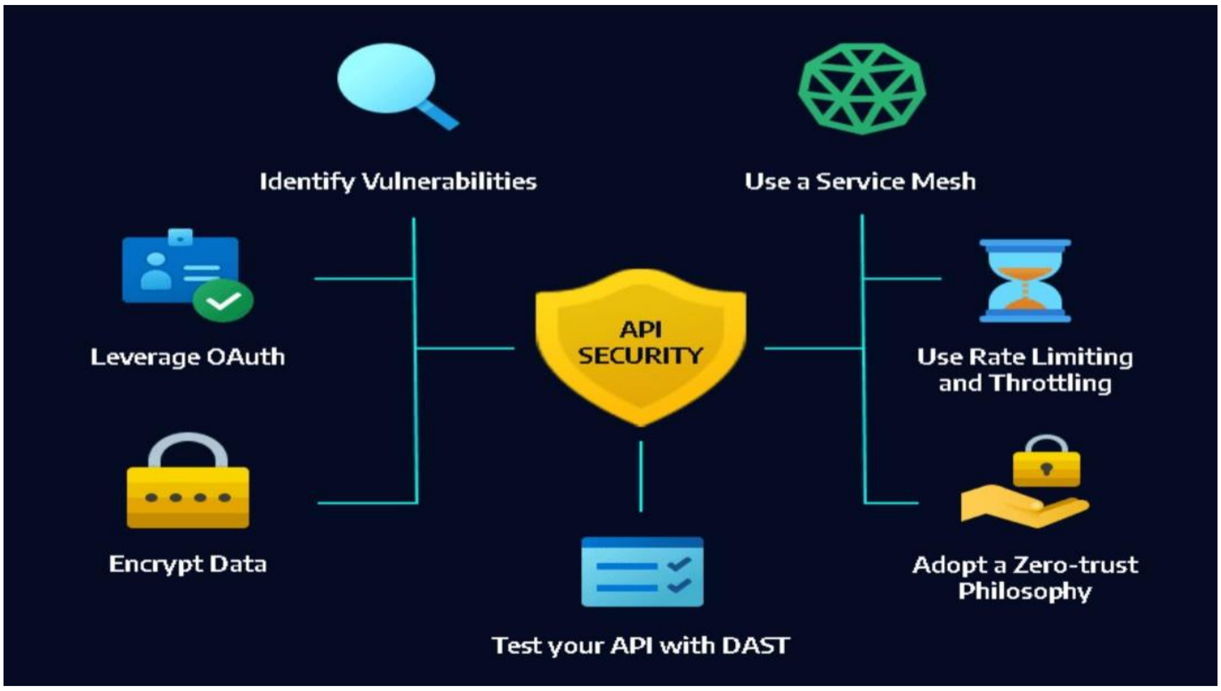
Zero Trust Flow

```
Service A → mTLS → Service B
Service B → JWT Validation → Allow / Deny
```

Security Highlights

- Mutual TLS between services
- No implicit trust inside network
- Every request verified

Diagram 8: Production-Grade API Security Stack



Layered Defense

```
Client
 ↓
WAF
 ↓
API Gateway
 ↓
Auth Server
 ↓
Application
 ↓
Database
 ↓
Monitoring & SIEM
```

Security Highlights

- Defense in depth
- Continuous monitoring
- Incident response readiness

📘 Deep Dive into HTTP & API Security

A Comprehensive, Production-Ready Guide for Modern Backend Systems

📌 About This Guide

This document provides a practical, real-world deep dive into HTTP, REST APIs, and API Security, covering:

- ➢ Secure HTTP usage
- ➢ Authentication & Authorization
- ➢ JWT & OAuth2
- ➢ Spring Security architecture
- ➢ Microservices & Zero Trust
- ➢ Production-grade best practices

- ➢ Target Audience: Backend Developers, Java/Spring Engineers, API Architects, Security-aware Engineers

1️⃣ Introduction to HTTP & APIs

🔷 **What is HTTP?**

HTTP (HyperText Transfer Protocol) is the foundation of communication on the web.

It follows a client–server model:

Client → Sends requests (Browser, Mobile App, Frontend)

Server → Returns responses (Web Server, API Backend)

🔷 **What are APIs?**

APIs (Application Programming Interfaces) allow different systems to communicate.

Most modern APIs are REST APIs, which use HTTP as the transport protocol.

🔷 Example: Basic HTTP Request

```
GET /api/users/123 HTTP/1.1
Host: api.example.com
Accept: application/json
```

Response

```
{
 "id": 123,
 "name": "John Doe",
 "email": john@example.com
}
```

2️⃣ Why HTTP & API Security Matters

🚨 The Stakes Are High

💰 Data breaches cost companies millions

🎯 APIs are top attack surfaces (OWASP API Top 10)

📜 Compliance requirements (GDPR, HIPAA, PCI-DSS)

🏷️ Brand trust & reputation damage

⚠️ Real-World Insecure API Example

GET /api/orders?user_id=456

❌ Returns all orders, including PII data, without authorization checks.

- Lesson: APIs must NEVER trust user input.

---

3 **HTTP Methods & Secure Usage**

Method	Purpose	Security Considerations
➢ GET	Retrieve data	HTTPS, rate limiting, no state change
➢ POST	Create resource	Input validation, CSRF protection
➢ PUT	Update resource	Idempotency, ownership validation
➢ DELETE	Remove resource	Confirmation, audit logging
➢ PATCH	Partial update	Prevent mass assignment

✅ Secure Spring Boot Example

```
@RestController
@RequestMapping("/api/products")
Public class ProductController {

  @GetMapping("/{id}")
  Public ResponseEntity<Product> getProduct(@PathVariable Long id) {
    If (!hasAccess(id)) {
      Return ResponseEntity.status(403).build();
    }
    Return ResponseEntity.ok(productService.findById(id));
  }

  @PostMapping
  @PreAuthorize("hasRole('ADMIN')")
  Public ResponseEntity<Product> createProduct(
      @Valid @RequestBody ProductDTO dto) {
    Return ResponseEntity.status(201)
        .body(productService.create(dto));
  }
}
```

4️⃣ **HTTPS & Transport Layer Security**

🔐 Why HTTPS Is Non-Negotiable

Encrypts data in transit

Prevents Man-In-The-Middle attacks

Authenticates server identity

Mandatory for modern browsers & APIs

✅ Secure Nginx TLS Configuration

```
Server {
  Listen 443 ssl http2;
  Ssl_protocols TLSv1.2 TLSv1.3;

  Add_header Strict-Transport-Security
  “max-age=31536000; includeSubDomains” always;
}
```

5️⃣ **Authentication vs Authorization**

Concept	Meaning
Authentication	Who you are
Authorization	What you can do

Common Authentication Patterns

1. ❌ Basic Auth (avoid in production)
2. 🔑 API Keys (server-to-server)
3. 🔐 OAuth 2.0 (industry standard)
4. 🎫 JWT (stateless tokens)

Typical JWT Flow

User → Login → JWT Issued

User → API + JWT → Verified → Data Returned

6️⃣ JWT Token Security (Deep Dive)

➢ **JWT Structure**

Header.Payload.Signature

✅ Secure JWT Implementation (Node.js)

```
Jwt.sign(payload, JWT_SECRET, {
  Algorithm: 'HS256',
  Issuer: 'your-api.com',
  Audience: 'your-app.com',
  expiresIn: '1h'
});
```

🔴 Never store passwords or secrets inside JWTs

7️⃣ Role-Based & Permission-Based Access Control

RBAC with Spring Security

```
.requestMatchers("/api/admin/**").hasRole("ADMIN")
.requestMatchers("/api/user/**").hasAnyRole("USER","ADMIN")
```

Method-Level Security

```
@PreAuthorize("hasRole('ADMIN') or #userId == authentication.principal.id")
```

---

8 CORS & Browser Security

- Why CORS Matters

Prevents malicious cross-origin requests

Protects authenticated browser sessions

Secure Spring Boot CORS Config

.allowedOrigins(https://trusted-domain.com)

.allowedMethods("GET","POST","PUT","DELETE")

.allowCredentials(true)

---

9 Input Validation & Rate Limiting

- FastAPI Validation Example

```
Class UserCreate(BaseModel):
  Username: constr(min_length=3)
  Email: EmailStr
```

Rate Limiting (Redis)

Depends(RateLimiter(times=5, seconds=60))

🔟 **Spring Security Architecture**

- **Core Components**

1. SecurityFilterChain
2. AuthenticationManager
3. UserDetailsService
4. PasswordEncoder
5. SecurityContext

Stateless JWT Configuration

```
.sessionManagement(
 Session -> session.sessionCreationPolicy(STATELESS)
)
```

1️⃣1️⃣ Microservices & Zero Trust Security

- ➢ Zero Trust Principles

Never trust, always verify

Least privilege

Assume breach

API Gateway + Service Mesh

API Gateway → Authentication, rate limiting

Service Mesh → mTLS, authorization policies

1️⃣2️⃣ Production-Grade Security Checklist

✅ API Security Checklist

OAuth2 / JWT

HTTPS + HSTS

Input validation

Rate limiting

Centralized logging

WAF & secrets management