Day 12 (MT, OauthOpen Id, sam)

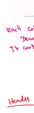
Authentication → verify identity
→ who are you?

Authorization → Access to resources
→ Can you access resource?

Pros cons
Easy to use
Can be used for web & mobile
Cons
→ cookie
→ CSRF
→ Proxy
→ Firewall

Problem with above:
Stateful
Scalability issue
Load balancing become complex
Microservices stateful by sticking Session ID

To solve above, came
Just → JSON web Token

→ stateless
→ web tokens


Demo: jwt.io

Black color encoded corresponds to Decoded color
JV contents/Header/
→ Header: { "alg": "HS256", "typ": "JWT" }
→ Payload: { "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }
→ Verify signature: base64UrlEncode(header) + "." + base64UrlEncode(payload), secret

Header, Payload, Signature

Header
{
"alg"
"typ"
"kid"
}

Header: { "alg": "HS256", "typ": "JWT" }
Payload: { "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }
Signature: HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

Header.Payload.Signature

Client
Server
Post login/(email,pwd)
Send Set Cookie
Cookie = Sign(Session id, secret)
Send req with Cookie
Send Response
Sign Check
Cookie = Sign(Session id, secret)

'Sub' itself has encoded format of user specific information so this key present only with Server with 'iat' & 'exp'

Authorization
Know what user allowed

To access Google sign-in for 3rd party
3rd party Company
Google Sign-in

→ get google sign-in id
Access to limited resources without exposing credentials for specific period of time
OAuth2.0

To solve this, use
OAuth2.0 Auth Server
Google Resource (Server)
User
Google Sign-in
Access service & consent & scope
Request Permission Scope
Request Approved
Resource Server
Authorization Code
Client id & Secret
Exchange Code for Access Token
Access Token
Access Resource (Protected Resource)
Access Resource (Protected Resource) for further
Access Token check
Return Resource

for multi-app needs on complex we need to login against system

To solve this
SSO → Single Sign-On

→ Access Multiple App/Services
→ Single Set of login credential
→ Share identity

SSO Client
Security Assertion Markup Language
→ open ID connect

In open ID, we combine previous concepts
Just
OAuth2.0 → Access Tokens is basically Just token

SAML actually uses XML

Client
Service provider (SSO login Page)
Identity provider
data store manager
→ Validate
→ Send SAR
→ Show login Page
→ login credential
→ Send SAML response
→ Send SAML response
→ Auth Response

Component1 Component2

How to know component 1 is alive?
safe way Heartbeats
helps detect fault in heartbeats early
heartbeat check
Availability check
Fault tolerance
Resource conservation

we can do this in "Service Discovery"

Architecture
In Just 5 mins summarize entire design
Why Quickly throw → Terms that we didn't go through in discussion
Auth, Monitoring, logging

Log Stack
→ Log Monitoring/Logging
ELK Stack
E: Elastic Search - searchable database
L: Log Stash - Transform Data (pipeline to transformation)
K: Kibana - Visualization Dashboards
ELK stack is for log/monitoring

CQRS
Command & Query Responsibility Segregation

Traditional
Presentation Logic (FE)
Business Logic (BE)
Persistence Logic (DB)

Client
CQRS
Query Handling (R)
(C/R)
FE
BE
DB

CRUD app
C/U/D - Command handling
R - Query handling
CQRS is helpful to optimize both Read & Write

Order service
C/U/D
Order Command Service
R
Order Query Service

Mock Review
Mock (Resume, behavioral) . . . .
CQRS on 21st Project, the Functional Consultancy

Mock Review Checklist → do HLD Journey
→ whatsapp vs Discord
→ Youtube vs Netflix vs Zoom
→ Uber, Zomato