Token based Auth

- After login, the server gives the client a token. client sends that token with every request to prove identity. Instead of remembering the user on the server (sessions), client carries proof of authentication.

* Traditional session-based authentication stores session data on the server.
- But when system scales horizontally, session storage becomes difficult.

Solution

Token based auth solves this by:
- Removing server side session storage.
- Allowing stateless APIs, making scaling easier

Working

- Login - user sends email+pass → server verifies credentials and generates a token & sends it back to client.
- Client stores token in memory, HttpOnly cookies.
- client sends token in header & server verifies & grants acce (No DB lookup if JWT).

Security Consideration

1) Always use HTTPS (token must never travel unencrypted)
2) Store tokens safely (Use HttpOnly secure cookie)
3) short expiry
4) Token revocation strategy.
   JWT is stateless. To revoke:
   - Maintain blacklist
   - Rotate signing keys

Note:- JWT is more scalable that session not automatically secure.

Access Control Lists & Rule Engines

- ACL is a list that defines who can access what & what actions they can perform.
- Suppose we have 100 users & 500 documents.
  Each document may have: Owner, editor, viewer.
  ACL would store rules like:
  - User 1 → read → doc 45
  - User 2 → edit → doc 45

* ACL is used in :-
  - File systems
  - Cloud storage (S3 bucket policies)
  - Database row-level security
  - Document management system.

Rule Engines: A system that evaluates logical conditions to decide whether something is allowed.

- Instead of storing static permissions, it evaluates rules dynamically.

* These are needed because modern APPs need :
  - Dynamic authorization
  - Policy-driven system.
  - Context based access

* Rule engines allow flexible policies, changing rules without changing code, scalable permission logic

Real world System

Large system combine :-
- Authentication + ACL + Rule Engine

Ex:- Banking system

ACL → User can access account
Rule engine → user can transfer only if bal > x.

Rate Limiting

- Restricting how many requests a user (or IP) can make in a certain time.

Why :- • Attackers can brute force password.
• user can overload your API.

It can be applied :
- Per IP address
- Per user ID
- Per API Key
- Per route

Algorithms

i) fixed window :- 100 requests / min
ii) sliding window :- counts requests in a moving time window
iii) Token bucket (common) :- • System gives tokens at fixed rate.
- Each request consumes one token.
- If no tokens left → blocked.

DDoS Protection

- Distributed Denial of service.
- Millions of machine flood your server with traffic at the same time to make your system unavailable.
- Rate limiting works at application level but DDoS attacks happen at network level.

Working

1) CDN (content delivery network) absorb traffic before it reaches your server.
2) Web App Firewall filters bot traffic, malicious pattern.
3) Load balancers distribute traffic across multiple servers.
4) Auto scaling (servers are added automatically)
5) Network level filtering (Block traffic at edge level).

Distributed Rate Limiting

* Enforcing rate limits consistently across multiple servers.
  All servers must share same request count.

* Need

Modern system scale horizontally, runs in containers & uses microservices.

Without DRL

- Limits can be bypassed, stability of system reduces.

- All servers must use shared central storage to track request counts.
- Instead of storing counters in memory, we store them in:
  - Redis
  - Database (less common)
  - Distributed cache

Why Redis is commonly used?

- Fast (in-memory), supports expiry (TTL) & lightweight.

Working :- 1. Request comes to server A
2. Server checks Redis → increment counter → check value
3. If value > limit → reject request

Note:- Since Redis is shared, all servers see the same count

Advanced concepts

Large systems may use:-

- Shared Redis clusters
- Edge level rate limiting (CDN)
- Adaptive rate limiting (based on load)
- Hierarchial rate limits.