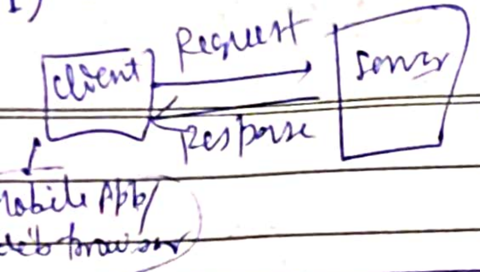
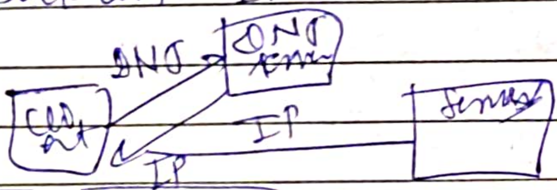
1) Client-server Architecture



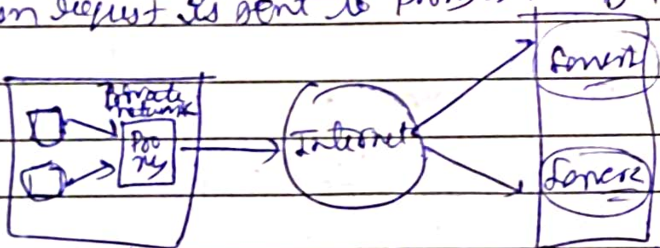
Client communicate with server for → store or retrieve data
- Create/post data
- Retrieve data
- Update data
- Delete data

2) How will client know server's address → Every machine on network is identified by IP address. It's tough to remember IP of machine, so need Domain name. & we need DNS server to map domain with IP.

3) DNS → Domain name system receives request of Domain Name & responds back with IP.



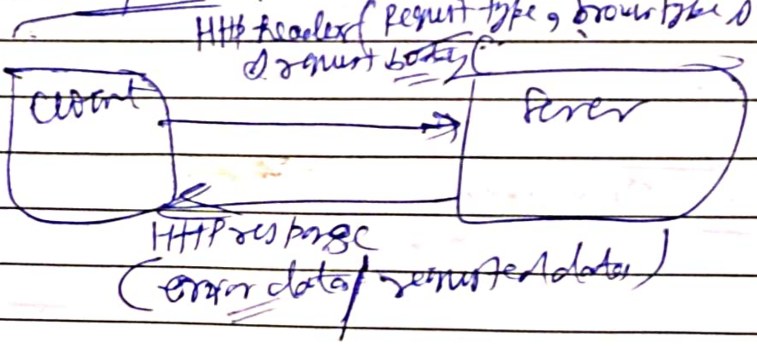
4) Proxy/Reverse proxy → Proxy server acts as middle man b/w device & Internet. When request is sent to proxy server & proxy server forwards request to server & receives response from server & returns back to client. It hides IP address, keeping location & identity private.



Reverse proxy → It intercepts the client request & forwards them to backend server.

5) Latency → Whenever client communicates to server, there is delay. It might happen if client & server are in different geographic location, so it takes time to travel data. Due to high network latency, can make application slow or unresponsive.

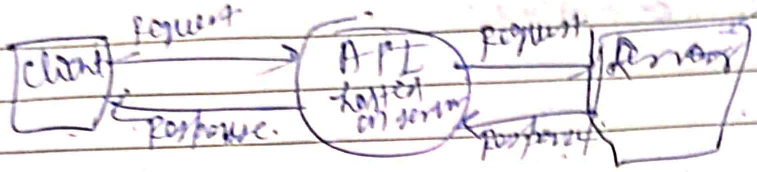
6) HTTP/HTTPS :- Client & server communicates using HTTP/HTTPS



HTTP sends data in plain text which can be attacked/modified so HTTPS should be used which provides encryption at TLS/SSL

7) HTTP does not define
- How request should be structured
- What format response should be
- How different clients should interact with server

↳ API (Application programming interface)



→ Client sends request to API and API sends request to database/services & receives response

→ API sends response in structured format (JSON/XML).

8) Different API Style
- REST
  - Stateless
  - Every request is independent
  - Everything is treated as resources (URLs/ends)
  - follows CRUD
- GraphQL

9) To store data, we need databases.

Database
- SQL →
  - Predefined schema
  - ACID properties
  - Strong consistency
  - Structured relationships
- NoSQL →
  - High scalability
  - performance
  - Flexible schema
  - → Key-value, Document, Graph.

Which one should be used
- Structured relational data + strong consistency → SQL
- High scalable + Flexible schema → NoSQL

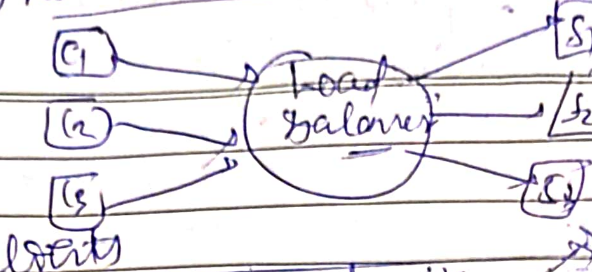
10) Vertical scaling → If more requests hit server then we need to upgrade server in terms of adding more CPU + RAM + storage

11) Horizontal scaling → Adding more machines → If one server is down other can be used so reliable

सी डैक CDAC

12) Load Balancer ← Servers

It is designed to Load server sits b/w clients & servers & works a traffic manager. If one server crashes, load balancer redirects traffic to other healthy server



Load balancer algorithms → Round robin, Least connections, IP hashing

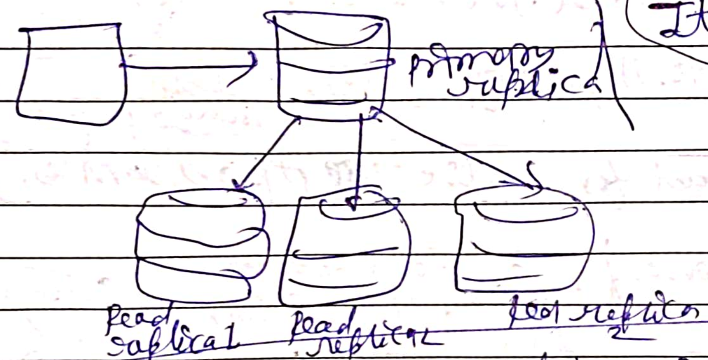
13) As traffic grows, the volume of data also increases. Other database scaling techniques →

Indexing → primary keys, foreign keys & to search data efficiently

But whenever we add data, need to update indexing so it should be done for more frequently accessed columns

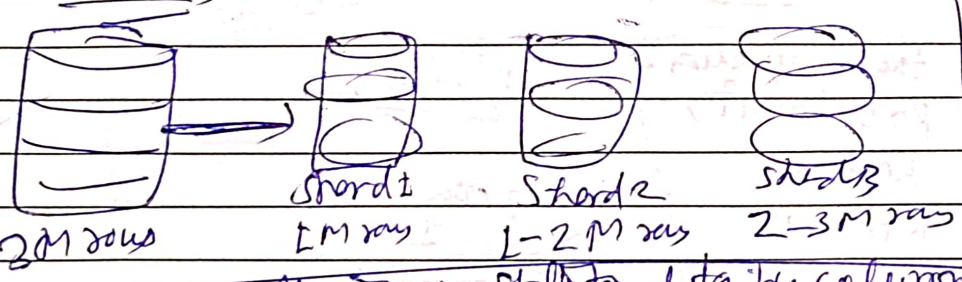
14) Replication → It improves reliability & performance & reduces

It is better for read heavy loads application

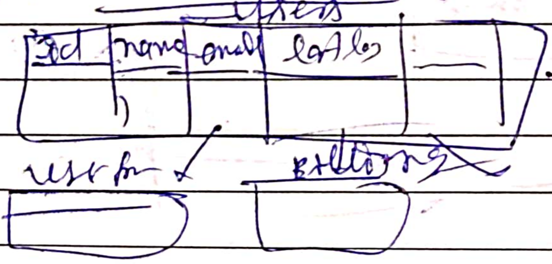


15) Sharding → divide database into smaller parts called shards

It is also called horizontal partitioning since it splits data by rows



16) Vertical partition → splits data by columns →

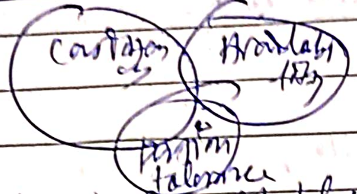


17) Caching → 

Application will be searched in cache if not found then after retrieval from DB, it is written to cache & then returned back to user request.

18) ~~Denormalization~~ Normalization

19) CAP



Since network failure is inevitable, Need to choose b/w C or A

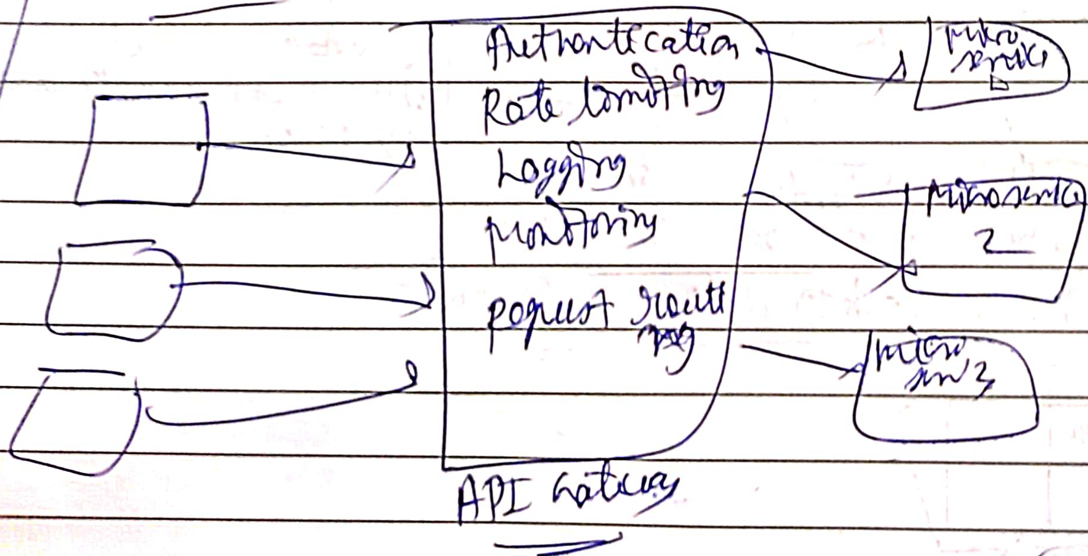
20) Blob storage → Amazon S_3 → to ~~store not only data~~ store data like audio/image/video or large files

Rate limiting → prevent overload for public ~~API~~ API & services, we need rate limiting.

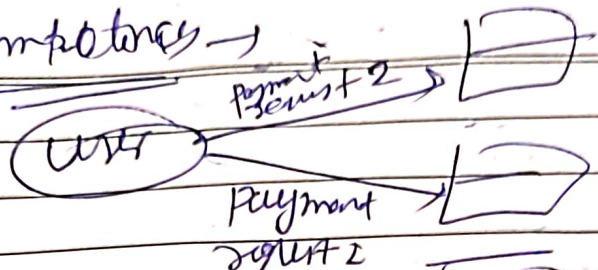
It may cause → crash of server, slow down speed

Rate limiting Algorithms
→ fixed window
→ sliding window
→ Token Bucket

API Gateway → centralized services



Idempotency →



When user refreshes the page & 2 payment request is made payment request save against with request Id so duplicate request can be ignored by server

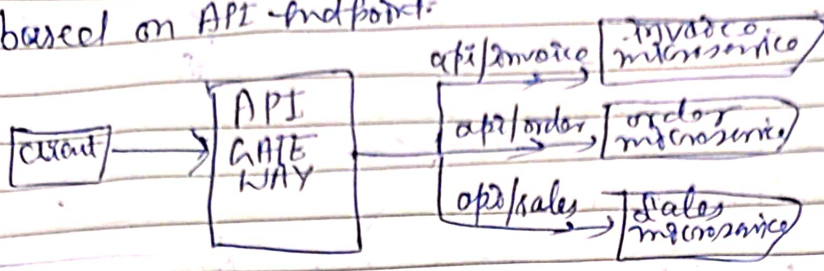
Normalization Vs Denormalization:

Normalization is the process of dividing data into multiple tables to reduce data redundancy & inconsistency. It increases no. of joins

Denormalization → It combines the tables/data so that it can be queried faster, as it does not require joins operation

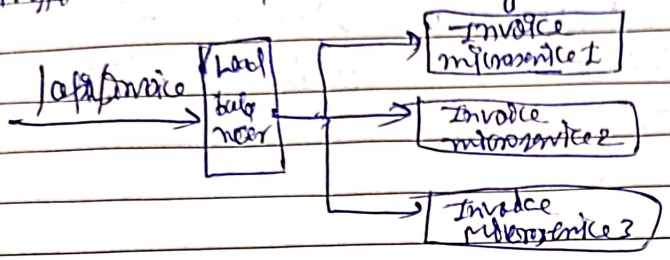
API Gateway →

It accepts the client API request & route them to correct backend service based on API endpoint.



Is it similar to load balancer → No, Load balancers simply distributes the traffic to multiple instances of microservice based on various factors (health, traffic, load)

Load balancer doesn't have capability to understand API & take decision



What API Gateway does:

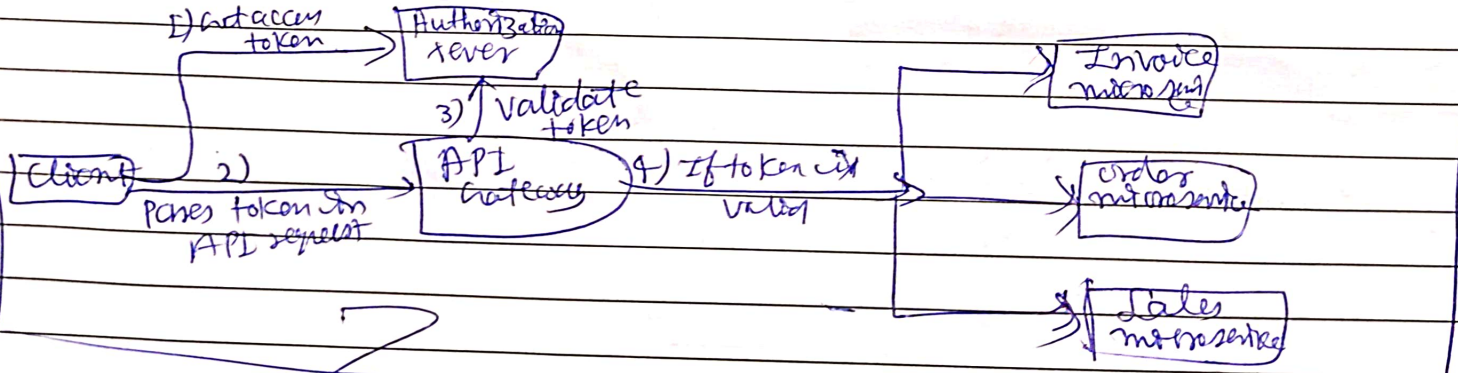
1) API Composition →

mobile device client → BFF → API to fetch product details, API to fetch invoice details

PC device client → BFF → API to fetch product details, API → Invoice details, Ratings & reviews, fetch recommendations

API gateway will call different relevant API on the basis of devices

2) Authentication



3) Rate limiting →
→ Manage burst limit → Set max no. of concurrent request to be accepted.
→ API Throttling → Limiting no. of request from individual or application
→ IP based blocking
→ API queues → Hold requests to an API.

4) Service discovery → As microservices can scale up & down, it's necessary to know the location of service: