Design an Online - Streaming Application Like HOTSTAR / NETFLIX

Step-1:- Requirement Gathering

① Functional Requirement

(a) User are able to search his Movie / Serial / Web Series etc.
(b) User able to add their favourite thing in watchlist.
(c) Get details of the serial / Movies / Web-Browser
(d) Watch History Maintenance
(e) User can give rating to movie / serial / webseries etc.
(f) Personalized Recommendations
(g) User can buy Premium Membership and can cancel Membership.
(h) User able to register / login.

OR

(a) Search Functionality
(b) Watchlist Management
(c) Detailed Information
(d) Watch History
(e) User Rating System
(f) Personalized Recommendations
(g) Buy / Cancel Premium Membership.
(h) Register / Login System.

② Non-Functional Requirement

① My System should be Highly Available.
② My System should be Highly reliable.
③ My system should be secure
④ My system should give high performance.
⑤ My System should be Scalable.

Note:- High Performance means Low latency Time.

Why High Performance or What High performance mean here in Online streaming Application:-

(a) streaming should be fast
(b) Instant Recommendation feature
(c) Smooth / Interactive User Interface
(d) System perform at regular speed even in large-scale traffic.
(e) Fast Response Time
(f) Immediate Action Execution
(g) No waiting Time.

Scalable:- App does not crash even User increase at large rate on App. Means if Daily Active User if increased by x% than normal numbers, my app must handle the load properly.

Reliable:- Means Minimizing the failures.

Step -2:- Prioritization

(a) Database Population:-

1.) **Primary Content Addition:** Data of Movies, Web series, TV shows are stored in Database.

2.) **Data Structuring:** Schema Design to check and track metadata, genres, actors, ratings and availability.

3.) **Dependency Resolution:** Our this content data will be foundation for search Recommendations and Subscriptions features. So their dependencies should be defined.

4.) **Scalability Consideration:-** Optimized storage and indexing is used in huge data maintenance and storage.

(b) User Management System:-

1.) **User Account Service:-** Maintain User Profile Data and their actions.

2.) **User Authentication Service:-** Authenticate User

login and registration

3.) **User Authorization Service** ← Optional
not much required (But if we want to make thing more systematic, we can add this feature).

(c.) **Search Feature**
Efficient content discovery is ensured in this step.

(d.) **Watch History and Favourite Content:-**
User engagement tracking here is ensured in this phase.

(e.) **Personalized Recommendation :-**
AI/ML-based suggestions are involved in this phase to build this feature.

(f.) **Premium Membership & Cancellation:-**
Monetization and user control features are ensured in this phase

In Prioritization, I focussed to build independent and most essential feature to build any application i.e from Top priority to the lowest Priority, i define my phase work

Step-3 :- Infrastructure Estimation / Back-of-the Envelop Calculation / Feature-in-Hand Planning

(a.) **Storage Requirements :-**

Let, Daily Active User = 1 Billion
Next year Estimated User = 0.5 Billion
Total users = 4 Billion
75% users are assumed to be inactive users in online

streaming application. (9+ is assumed)

Assume data of user be 1 MB

Total Data for user in Next year

$= 4.5 \times 10^9 \times 10^6$ Byte

$= 4.5 \times 10^{15}$ Byte

$= 4.5$ PB

Data for content :-

Let number of movies (expected upto next year) on app be 200000. Let 3GB on on average each movie must contain the data.

Estimated Data for Movie $= 3\text{GB} \times 2 \times 10^5$

$= 3 \times 10^9 \times 2 \times 10^5$ Bytes

$= 6 \times 10^{14}$ Bytes

$= 600 \times 10^{12}$ Bytes

$= 600$ TB [$\because$ 1TB $= 10^{12}$ Bytes]

Let, number of serials (expected upto next year) on app be 10,000. And each serial on an average consist of 750 episodes then

Estimated Data for serial $= 750 \times 10^4 \times 10^9$

$= 7.5 \times 10^{15}$ Bytes

$= 7.5$ PB

[$\because$ 1 PB $= 10^{15}$ Bytes]

Note:- Average size of each episode is taken as 1GB i.e of 45 minutes. each on average

Now, Let Web series upto next year be 5000, each web series of each episode containing content for 60 minute on an average on an average each web series contain 10 episodes each web series contain 3 seasons on an average

Each Web series Size $= 30 \times 2\text{GB} = 60$ GB per Web series

Total data required for Web series $= 60 \times 10^9 \times 5 \times 10^3$

$= 3 \times 10^{14}$ Bytes $= 300$ TB

Final Approximate Storage Estimation

For Data Content Estimation for storing video :-

8.5PB (7.5PB + 0.6TB + 0.3PB)

But, we apply some compression algorithm like lossless decomposition that reduce data need by 20% easily without Quality loss.

Final storage requirement for storing content

= 80% of 8.5PB

= 0.8 × 8.5 PB

= 6.8PB (approx)

≈ 7PB

for new content, we have taken 1PB as buffer storage

So, Total data for Distributed File System = 8PB

Metadata storage Estimation :-

Let 1MB is required to store basic detail about movie, serial, web series i.e :-

① Name ② Release / Ending Date

③ Actor Cast ④ Producer / Director information

⑤ Channel Detail (for Serial)

⑥ Link to the path where content is stored

For Movies :- 200K × 1MB = 200GB

For TV serials :- 10K × 1MB = 10GB

For Web Series :- 5K × 1MB = 5GB

Total Metadata Storage Required = (200 + 10 + 5) GB

= 215 GB

Caching & Extra Indexing can take upto extra data of 35GB.

So, Total Metadata Storage Required = 250GB

Data for storing User interaction & Activity logs for storing Watch History, Search Queries, Click events, personalized recommendations etc.

Estimation :-

Daily Active Users : 1 Billion
Every User :- 10KB /day activity generation
Daily storage :- 1 Billion x 10KB = 10TB /day
Annual Storage :- 10TB x 365 = 3.65PB /year
Data for Cache storage (Fast Access data)
10-20% of total content data is stored in cache on an average.
Now, Let 20% of data (Total data) is stored in cache.
Rough estimate :- 2PB for caching

Final Data Storage Summary :-

User Data :- 4.5PB
Content Storage :- 10PB
Meta data Storage :- 215GB (~0.2TB)
User Activity logs :- 3.65 PB / year
Cache Storage :- 2PB
Total Storage :- 20.35PB (including all components)

Let, Total Storage :- 21PB
As Buffer for future growth ~ 1PB
Which is good for our system to be made scalable.

② Traffic Estimation

Major factors to measure it :-

(a) DAU (Daily Active Users) → 1 Billion
(b) Peak vs Avg. Load
(c) Requests per second
(d) Read vs Write Ratio
(e) Concurrent Users (f) Session Duration

Traffic Estimation is Basically estimating the load of upcoming requests to our system.

(b) Peak vs Avg Load Estimation :-

Let 1 hour/day is average time for each Daily active user for streaming.

1 hour = 3600 seconds

1 Billion Users / 24 hours = 41.6 Million Users per hour

41.6 Million / 3600 sec = 11500 users per second

Avg Load = 11500 Users per second

Peak Load = 4 times Avg Load

= 4 x 11500 Users per second

= 46000 Users per Second.

Request per second depend upon :-

(a) Homepage requests (Searching, Recommendations)

(b) Video Requests (streaming, Buffering)

(c) Database Queries (Watch History, Favourities)

Using Traffic Estimation to find :-

(a) Server Capacity Plan

(b) Scalability Planning

(c) Caching Strategy

Concurrent User & RPS (Request per second) Load Estimation :-

Peak concurrency Factor ≈ 1 to 5% of DAU

for 1 Billion DAU, Let 1% be the peak concurrency factor i.e 10million users.

Let, Each User do 5 operations per minute

So, Per Minute Request = 10M x 5 Request

= 50M Request per minute

Per Second Request or RPS = $\frac{50M}{60}$

= ~ 833K

Estimated RPS for Worst Case

Read Vs. Write Ratio

By Analysis, Application will be ready-heavy.

Major Read Operations :-

① Homepage Load :- Recommendations, Trending Videos
② Search Query ④ Watch History Fetch
③ Streaming Request ⑤ Favorite List Fetch.

Minor Write Operations :-

① Give Rating ② Favorite List add/delete
③ Updating Watch History

Estimated Ratio: 95% Reads & 5% Writes

Throughput Calculation :-

Throughput means how much request a system can successfull process in a second.

Now,

Estimated RPS = 833K

Throughput = RPS × Average Request Size.

Let Avg Request Size = 10KB

Then, Throughput = 833K × 10KB
= 8.3GB/sec.

If request is of video stream then request size can also be longer

Bandwidth estimation :-

Mostly used for video streaming estimation

In streaming, Bit rate matters a lot i.e

Video Quality	Bitrate (Mbps)	Size per hour GB
480p (SD)	1Mbps	0.45GB
720p (HD)	3Mbps	1.35GB
1080p (Full HD)	5Mbps	2.25GB
4K (Ultra HD)	20Mbps	9GB

Now, If 10 million users are streaming at avg 1080p quality (5Mbps)

Total Bandwidth required = 10M × 5Mbps
= 50Tbps (Terabits per second)

The analysis done for Bandwidth Estimation is for Worst-Case Analysis But in Reality by CDN i.e Content Delivery, Network bandwidth requirement can be reduced 10-20 times.

Infrastructure Estimation Steps:- (Followed)

① Request Handling:-
(a) Calculating RPS
(b) Estimated 100K CPUs to process 100 RPS per core (Theoritically a High End Machine in an ideal case of RAM 16-32 GB can handle 8.3 GB/sec Throughput single handedly, but in practicality it is not possible)
(c) Autoscaling required for peak loads
(d) Video Processing (Encoding / Transcoding)
→ GPU-based Cloud instances needed
→ Parallel Processing to optimize Encoding Speed

② Network Bandwidth Estimation:-
(a) 50Tbps for peak load
(b) CDN & Edge Caching to reduce direct load.
(c) 8.3 GB/sec → Throughput Calculation for API requests & User Actions.

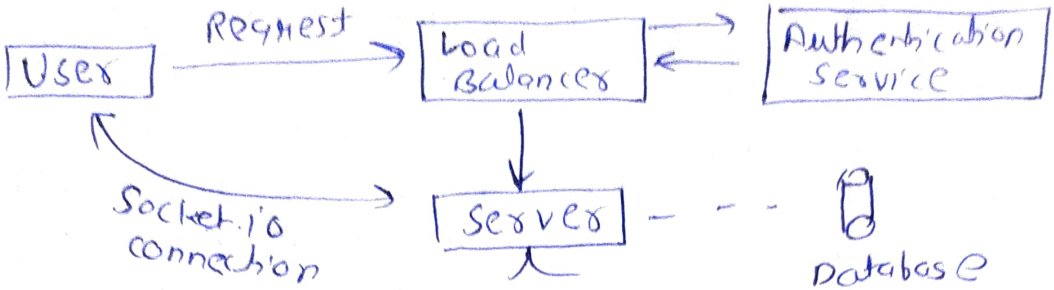
③ Scaling Strategy:-
Database Sharding, Horizontal scaling (Microservices + Load Balancers), Auto Scaling (Real-Time Resource Optimization based on Load), Vertical Scaling (Efficient Resource Allocation on cloud instances), CDN & Edge Computing (Reduce global latency for video delivery).

④ Traffic Estimation:-
→ Estimate Daily Active User → 1 Billion
→ Estimate Concurrent Users at Peak time → 10M
→ Find our Application is Read-heavy or Write-heavy. → Read Heavy

In this step, Infrastructure Estimation is done.

Step-4:- Component Identification

(a) Starting with very basic design:-



Introducing Zookeeper so as to choose right server as it monitors health check-ups Maintain Zookeeper in distributed manner which are given some servers in which they send their request by their Health Monitor checkup

More Optimization:-

Active / Passive LB → Distributed zookeeper system

First depending on Load, Algorithm used in LB will be:-

(a) Round Robin Algorithm
(b) Least Connection Algorithm
(c) Weighted Round Robin Algorithm

↓ Request size Tses

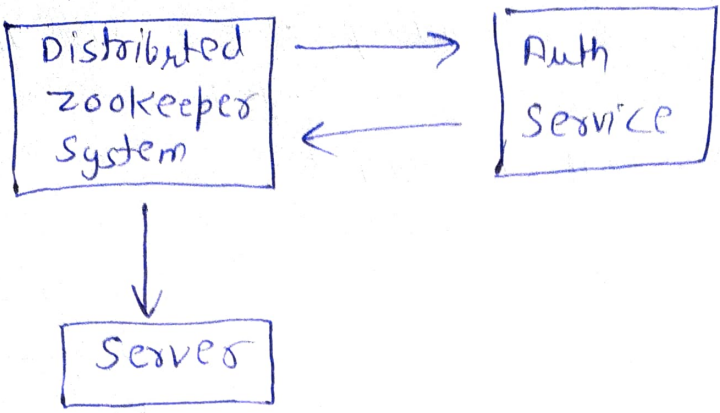
Advantage of this Design:-

(a) High Availability:- LB & Zookeeper both handle failure.

(b) Scalability:- Round Robin approach followed by Least Connection & Weighted Round Robin Algorithm if load Tses) evenly distribute load.

(c) Fault Tolerance:- If Zookeeper failed, Backup instance will handle it.

Now,



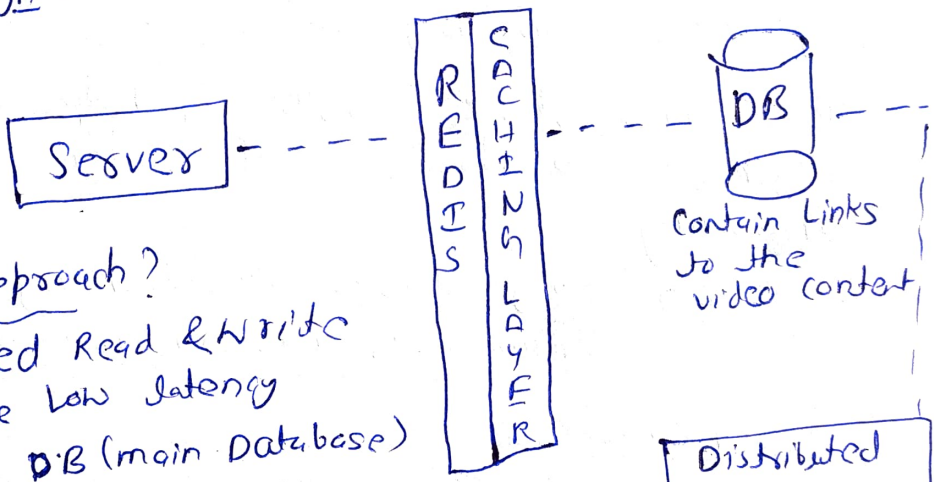
Advantage of this Approach :-

(a) Secure Authentication (b) Efficient communication

(c) Low Latency
(Socket.io bring low latency to the system)

Zookeeper exit itself when communication is established b/w user and server, this optimizes the load

More Design :-



Why this Approach?

① High speed Read & Write
↳ Provide Low latency
② Reduced DB (main Database) load.
③ Better Scalability, Multible server can connect / efficiently scaled easily.

Server taken here is basically a Distributed server system

Note:- Video is streamed on server to user app in the form of small video packets. We can use CDN or Web socket connection for it

Load Balancing Chosen → L4 LB

Why chosen L4 Software LB?

(a) Cost Effective (b) Faster Routing due to TCP/UDP
(c) Better Scalability (Horizontal Scaling Possible)
(d) High Performance (High Throughput

More Design :-

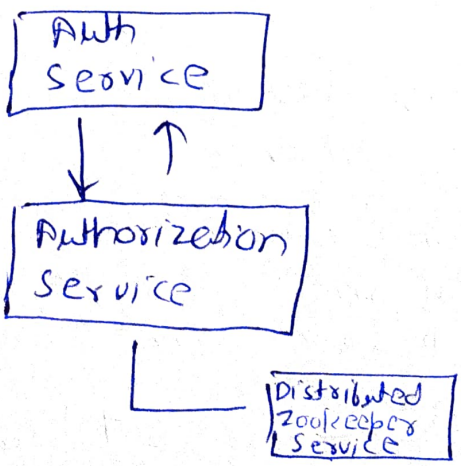
We need to store 4.5PB of data which is very large. Due to this we prefer PostgreSQL as it's scalability, performance & reliability is much higher than ~~BCG~~ MySQL. So PostgreSQL is preferred for Big Data Workloads

You connot store 4.5PB of Data in single machine. To store it efficiently, we use Sharding

↳ Range-based Partitioning
(Each User has Unique ID)

Advantage

① Linear scalability ③ Fault Tolerance
② Better Performance ④ Write Optimization



Main DB is in Cassendra Data Type.
And Redis is used where caching is used
User activity logs of 3.65 PB data can be stored in Cassendra

Why Cassendra :-

(a) Scalability :- It is horizontally scalable, can efficiently handle huge datasets.

(b) High Write Throughput :- 5-16 PB of data can be efficiently stored in Cassendra. For storing 16PB of data store, high write speed is required that is efficiently managed by Cassandra log-structured storage Engine.

(c) No Single Point of Faidure :- Cassandra use Peer to Peer Architecture so that system will not crash if a node fails.

(d) Range Queries :- (Partition key + Clustering Columns) can be used for optimizing user-specific queries.

(e) Time-Series Data Support :- If video access logs or user activity store is required then Cassandra is optimized for time-series data also.

In Cassandra DB, Shording can be done to increase DB query optimization.

Caching layer is put over the Cassandra Database.

ID Generator Service :-

Produce unique ID based on the content given to it.

To produce Unique id for User, transactions etc.

Since Our application is read-heavy. Master-Distributed DB Architecture is preferred.

Strategies for Distributed Database :-

(a) Master-Slave For User-DB because of high read operations
(b) Master-Master for high-write components like for user generated content like comments, reviews etc.

Stateless or Stateful System:—

Stateless is prefered because:—

(a) Scalability :- stateless are easily horizontally scale.

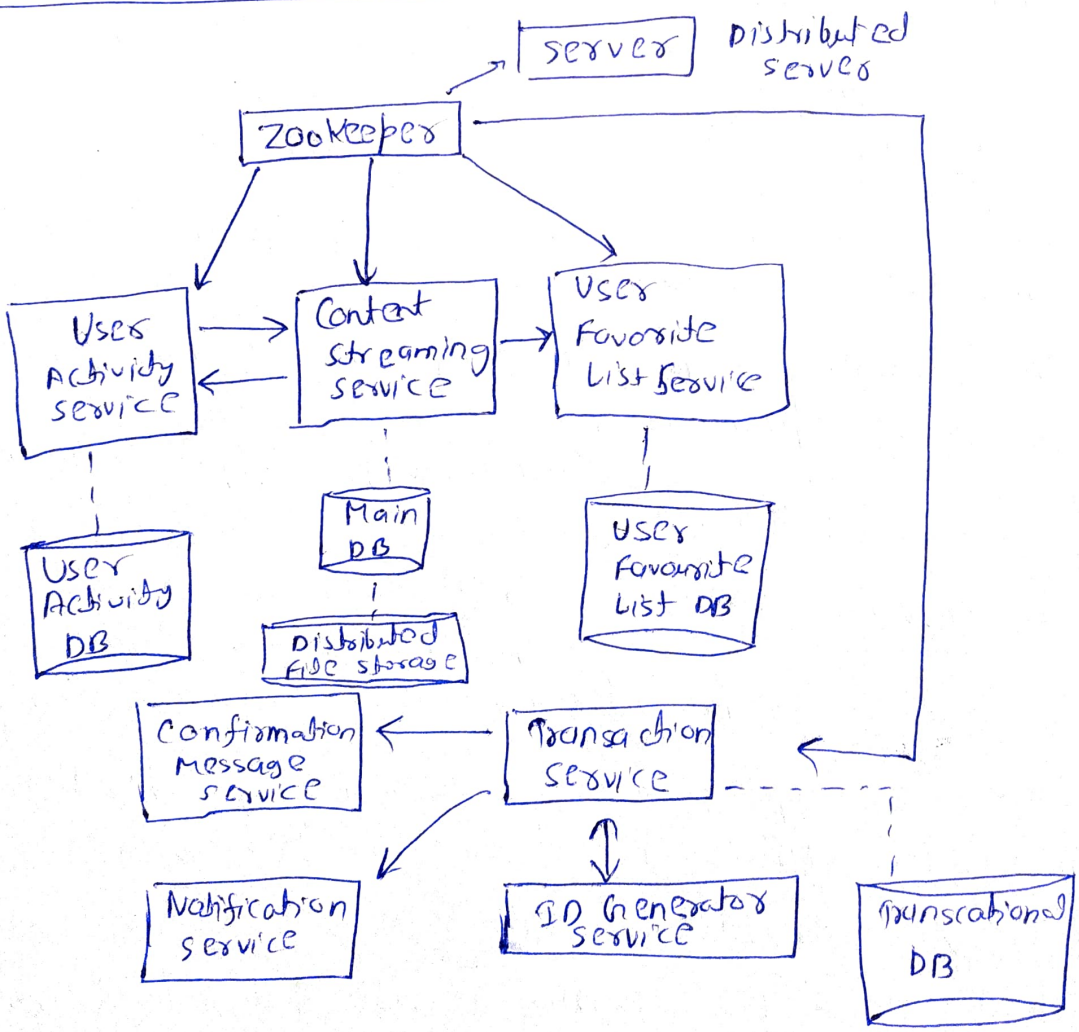
(b) Microservices Architecture:- Best, because services works independent of each other in it.

(c) Load Balancing:- Can send request to another system as no session dependency will be there.

(d) Failure Handling:- If a server failed, request go to another server without any data loss.

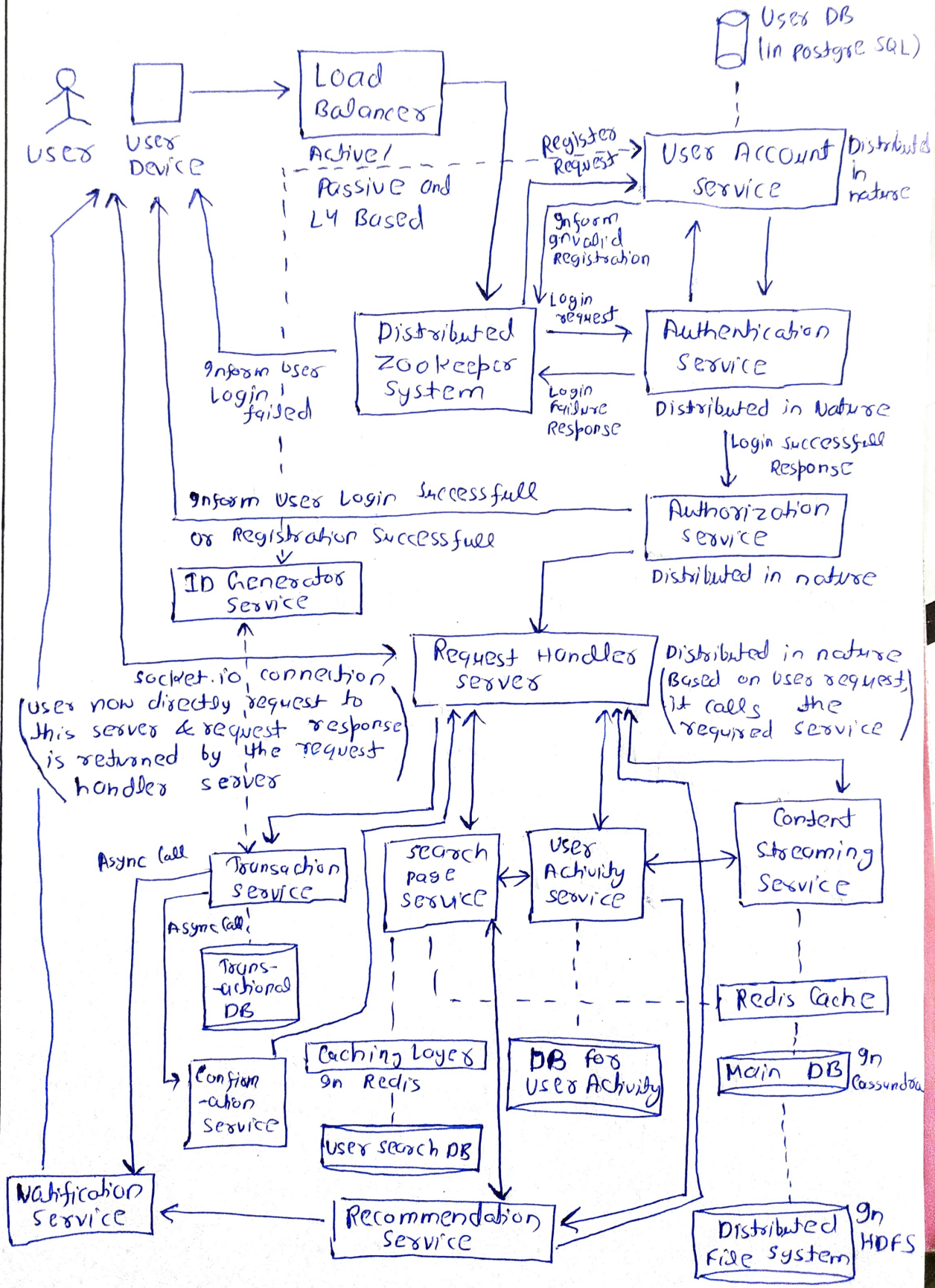
(e) Service to service communication via REST (JSON over HTTP)

Introducing More Service to Complete Our All Functional And Non Functional Requirement



Search Page service and Recommendation service are also required.

step-5 :- Final Design and Communication Protocol Identification



Full Communication Protocal :-

From	To	Protocol	Reason
User Device	Load Balancer	REST (HTTPS)	secure Request Routing.
Load Balancer	Request Handler service	REST (HTTP)	Request Forwarding
Request Handler service	Authen-tication service	REST (HTTP)	User Login Authentication
Authent-ication service	User Account service	SQL Query (Postgre SQL)	User details Fetch.
Authenti-cation service	Distributed zookeeper system	Zookeeper API (TCP)	Distributed Instance Coordination
Authentic-ation service	Authori-zation service	REST (HTTP)	Role-based access control.
Authoriz-ation service	Request Handler service	REST (HTTP)	Access Approval
User Device	Request Handler service	Web socket	Real-Time Updates (login session, etc.)
Request Handler service	Trans-action service	REST (HTTPS)	secure Payments
Transaction service	Transa-ctional Database	SQL Query (Postgre SQL)	Payment Records
Transaction service	Notific-ation Service	Async Messaging (Kafka / Rabbit MQ)	Payment Confiomation Notification.
User Device	search page service	REST (HTTPS)	search Queries
search page service	Caching Layer (Redis)	Redis Binary Protocol	Fast search Results
search page service	User search DB	SQL Query (Postgre SQL)	search History
User Device	User Activity service	Web socket	Real-Time User Tracking
User Activity service	User Activity Database	SQL / NoSQL Query (Cassandra)	User actions Log

Full Communication Protocol :-

From	To	Protocol	Reason
User Activity Service	Recommendation Service	REST (HTTP)	Behavior data sharing
Recommendation service	DB for user Activity	SQL / NoSQL Query (Cassandra)	Analysis input
Recommendation Service	Notification Service	Async Messaging (Kafka / Rabbit MQ)	Personalized Recommendations
User Device	Content Streaming Service	REST (HTTPS)	Video Requests
Content streaming Service	Redis Cache	Redis Binary Protocol	Quick Video Loading
Content streaming service	Main Database (Cassandra)	SQL / NoSQL Query	Video Metadata.
Content streaming service	Distributed File storage (HDFS)	HDFS API (HTTP/TCP)	High volume Video storage
Content streaming service	User Device	UDP	High speed Video Streaming
User Device	Notification Service	Websocket / Firebase Cloud Messaging (FCM)	Real-Time Push Notification
Notification service	Database for Notifications	NoSQL (Cassandra)	Notification logs storage
Distributed Zookeeper System	All Distributed Services	Zookeeper API (TCP)	Cluster Coordination