Design an Online Food Ordering and Delivery System like Zomato / Swiggy

Step-1 :- Requirement Gathering :-

(a) Functional Requirement

(i) For Users

1. User can register / login
2. User can order Food.
3. User can cancel an order
4. User able to search his / her food
5. User can see Order History
6. User can see order status
7. User can add / remove food from his Wishlist
8. User can manage his / her cart service like add / remove food from his cart.
9. Personalized Recommendations.
10. Restaurant Listings & Ratings - Browsing the User restaurant to check reviews and give ratings.
11. Support & Feedback :- User can register for complaint or support.
12. User can see order arrival time (ETA).
13. User can update his/her profile.

(ii) For Delivery Agents

1. Get Notification of new order.
2. Accept / Reject Booking
3. Update Location frequently
4. Details of order and location checking
5. Delivery Agent can check delivery history.
6. Delivery Agent can update its profile

(iii) For Restaurants

1. Restaurant Profile Management
   - → Updating Name, Address, Contact Information of Restaurant.
   - → Setting Opening and Closing Time.

(2) Menu Management
→ Add / Edit / Delete Food Items
→ Updating Price
→ Food Availability control (Marking Food Out of stock etc)
(3) Order Management
→ Accept / reject incoming orders.
→ Setting Order Preparation time.
→ Sending Order Ready Notification to Delivery Agent
(4) Live Order Tracking & Status Updates
→ Real-Time tracking access
→ Which Delivery agent will pick up the orders, its details are also accessed.
(5) Payments & settlements
→ Daily / Weekly Payout Reports
→ Accessing Transaction History & Pending payments.
(6) Ratings & Feedback Analysis
→ Seeing or Accessing User Reviews & Ratings
→ Analysis of Negative Feedbacks.
(7) Promotions & offers (Optional)
→ Managing Discount Coupons and Festival Offers.
→ Partner based deals making with Online food ordering and Delivery system like Zomato.

Step-2:- Prioritization :-

(1) Data Population & Setup (Core Foundation)
(a) Primary Data Addition:- Restaurants, Menus, Locations
(b) Data Structuring & Dependency Resolution
(c) ID Generator SetUP :- Unique ID for Users, Orders, Restaurants.
(2) Authentication & Authorization Services (Security First)
Why :-
(a) Users, Restaurants & Delivery Agents are ensured secure access
(b) Authentication :- Managing Login / Signup
(c) Authorization :- Define Access Rights.

What will be Done?

(a) User Authentication service :- Signup / Login
(b) Token-based Authorization :- JWT, OAuth etc.
(c) Roles & Permissions set up :- User, Restaurants and Delivery Agent.

③ User Account service (Profile Management)

Why?

→ Profile set up of User is compulsory to place orders.
→ store User Preferences and Addresses.

What will be Done?

(a) User Registration & Profile Management
(b) Saved Addresses & Contact Details.
(c) User Preferences & Favourite Cuisines Tracking.

④ Restaurant Management Service (Manage Food Providers)

Why?

→ Without Restaurants, Core business of Online Food ordering and delivery system cannot exist.
→ Before order system, Restaurant setup is compulsory

What will be done?

→ Restaurant Profile Management :- Name, Address, Timings etc.
→ Menu Management :- Food items Add / Edit / Delete, Price update.
→ Order Processing System :- Accept / Reject Orders.

⑤ Menu & Inventory service (Food Availability Tracking)

Why?

→ The placing of order, it's backend will be handled in this service.
→ For Handling Out-of-Stocks items or unavailable restaurants, it is necessary.

What will be done?

→ Restaurant-Wise Menu storage
→ Real-Time Inventory Check
→ Dynamic Price Updates

⑥ Delivery Management Service (Assign & Track Delivery Agents)

Why ?
→ Actual Fulfillment of Order is not possible without Delivery.
→ Without Delivery Agents, Our system will remain incomplete.

What will be done?
→ New Order Notifications for Delivery Agents.
→ Accept / Reject Booking Options.
→ Live Location Tracking & status Updates

⑦ Search & Filtering Service (Find the Best Restaurants & Food)

Why ?
→ Major Part of User Experience
→ Fast & Relevant Suggestions should be available to User.

What will be done?
→ Text-Based Search :- Restaurant / Food Name
→ Filters :- Ratings, Distance, Price, Cuisine Type.
→ Sorting :- Best Rated, Fastest Delivery, Most Popular

⑧ Cart & Payment Service (Secure Order Placement)

Why?
→ Order Finalization and without payment service, it is incomplete
→ Payment Service is necessary for consistency and fraud detection.

What will be done?
→ Add / Remove items from Cart.
→ Apply Coupons & Discounts.
→ Secure Payment Gateway Integration.

⑨ Location Service Manager (Live Order Tracking & ETA Calculation)

Why ?
→ Real-Time Location updates improve Delivery Experience.
→ Important to calculate distance between User and delivery Agents.

What will be done?
→ Calculate distance between User and Restaurant.
→ Real-Time ETA Estimation.

→ Delivery Agent live Tracking System.

(10) Booking & Order Confirmation (Processing and Assigning Deliveries)

Why?
→ On order Confirmation, system need to fulfill it.
→ Restaurant + Delivery Agent coordination is compulsory.

What will be done?
→ Final Order Placement & Confirmation.
→ Assigning Restaurant & Delivery Agent.
→ Order Processing Workflow Execution.

(11) Notification Service (Real-Time Alerts & Updates)

Why?
→ Users, Delivery Agents and Restaurants should timely get updates.
→ There will be multiple stages after payment confirmation.

What will be done?
→ User:- Order confirmation, Delivery Updates
→ Delivery Agent:- New orders & Pickup Alerts.
→ Restaurants:- New order Alert and Ready-for-Pickup Notification

(12) User Feature Expansion (Advanced Enhancement)

Why?
→ Since, Core system is now stable, so add advanced features.

What will be done?
→ Order Arrival Time Estimation
→ Order Cancellation & Refunds
→ Order Status Updates
→ Wishlist & Favourite Items.
→ Ratings & Feedback System.

(13) Personalized Experience & Support (Enhancing User Experience)

Why?
→ User Retention and Satisfaction is essential.

What will be done?

→ AI-Based Personalized Recommendations
→ Customer Support & Feedback Feedback System.
→ Loyalty Programs and special Offers.

Step-3 :- Infrastructure Estimation & feature-in-hand Planning

(A) Storage Requirements :-

(a) For User :-

Let, Total Users on Zomato = 1 Billion
Daily Active User = 10% of Total Users
= 0.1 Billion

Now, Upto Next Year, new User registered
= 0.2 Billion (Assumed 20% growth)

Let, User data take approximately 20MB for each user.

20MB Breakdown Per User :-

(a) User Profile data :- 1 MB
(b) Order History :- 10MB
(c) Wishlist & Preferences :- 3MB
(d) search and Analytics :- 2MB
(e) Misc (Reviews, Ratings etc.) :- 4MB

Now, Total Storage required for User Account service = $1.2 \times 10^9 \times 1 \times 10^6$ (Upto Next Year)

$= 1.2 \times 10^{15}$ Bytes
$= 1.2$ PB $[10^{15}$ Bytes = 1 PB$]$

This data is stored in SQL dB and User Account service is connected to it. This dB contains the User profile data for Authentication & Authorization purposes.

By lossless decomposition 30% of data can be saved by this lossless decomposition

So, Total Storage Required in SQL DB
$= 0.7 \times 1.2$ PB
$= 0.84$ PB or 840TB

Order history DB data Estimation upto the next year = 1.2 Billion × 10 MB

$= 1.2 \times 10^9 \times 10 \times 10^6$ Bytes

$= 12 \times 10^{15}$ Bytes

= 12 PB

Cassandra DB is used here. which provide LZ4 compression by which 50% compression can be done easily without any data loss.

So, Data required will be 0.5×12 PB

Hence, Order History DB data = 6 PB

Why LZ4 Compression Algorithm :-

Fastest Performance, Real-Time queries.

Wishlist & Preference DB data = 1.2 Billion × 3 MB

$= 1.2 \times 10^9 \times 3 \times 10^6$ Bytes

$= 3.6 \times 10^{15}$ Byte

= 3.6 PB

Wishlist & Preference DB data after LZ4 compression Algorithm = 0.5×3.6 PB

= 1.8 PB

Elastic search (Document DB) for searching will be a good choice & Cassandra for Analytics will be good choice

Data for search & Analytics required to be stored = $1.2 \times 10^9 \times 2 \times 10^6$ Bytes

= 2.4 PB

search Data ≈ 1 PB —LZ4 Compression (50%)→ 0.5 PB

Analytics Data ≈ 1.4 PB —LZ4 Compression (50%)→ 0.7 PB

So, Overall Data for storing search & Analytics DB will be 1.2 PB

Now, Data required for storing miscellaneous data like reviews & rating = $1.2 \times 10^9 \times 4 \times 10^6$ (Upto next year)

$= 4.8 \times 10^{15}$ Bytes

= 4.8 PB

Database preferred here is Document DB, say MongoDB

40% compression can easily be applied by Zstd i.e Z Standard Compression algorithm without any data loss.

Hence, Total data required to store reviews/ratings (misc data) upto next year

$= 0.4 \times 4.8$

$= 1.92$ PB

Hence, Overall storage required for User

$= 1.92 + 0.84 + 6 + 1.8 + 1.2 +$ Redis (Cache Data)

$= 11.76$ PB + Redis (Cache Data)

Cache Layer Decision

(a) On order History :- 3 MB out of 10 MB due to Recent order Fetching

(b) Wishlist & Preferences (3 MB) → Full Caching
Wishlist is frequently modified so read-heavy

(c) Search & Analytics (2 MB) → Full Caching
search result must be very fast.

(d) Misc (2 MB out of 4 MB) i.e partial Caching :-
Recent reviews are frequently read. Last 100 reviews in cache

Total Cache storage per user $= 3 + 2 + 2 + 3$

$= 10$ MB

For all Users upto next year, Cache storage required

$= 10 \times 1.2 \times 10^6 \times 10^9$

$= 12$ PB

Compression Technique in Cache layer :-

(a) Snappy Compression for search, Analytics & Wishlist (50%)
i.e (2 MB + 3 MB) → Best for speed & Low Latency

(b) LZ4 (Low Latency & High Throughput) → for order History i.e 3 MB → (60% Compression)

(c) Zstd (Best Compression Ratio & Fast Performance) for Misc Data i.e 2 MB. → (80% compression)

Hence, After Compression Redis Data required =

$= (5 \times 0.5 + 3 \times 0.4 + 0.2 \times 2)\ PB$

$= (2.5 + 1.2 + 0.4)\ PB$

$= 4.1\ PB$

So, Cache Data (Redis) for User = 4.1 PB

(b.) For Delivery Agent :-

Let, Total Delivery Agent (Daily) = 62.5 M

Now, Let New Delivery Agent that do registration in the next year = 10% of DADA (Daily Active Delivery Agents)

$= 0.1 \times 62.5$

$= 6.25\ M$

Total Delivery Agent Upto Next Year $= (6.25 + 0.625)\ M$

$= 68.75\ M$

Let, 80% of Delivery Agents are active as Most of the Delivery Agents are Available in Online Delivery System.

Daily Active Delivery Agents = 80% of 62.5M

= 50 Million

Upto Next Year Daily Active delivery Agent

= 80% of 68.75 M

= 55 Million

Now, On Current Year Order Per Delivery Agent

$= \frac{1.0B}{50M}$

= 20 Orders/day

Now, On Next Year Order Per delivery Agent

$= \frac{1.2B}{55M}$

$= 21.8$ Orders/day

$\simeq 22$ Orders/day

Total data for Delivery Agent (Each Delivery Agent) = 15MB

15MB Breakdown Per Delivery Agent :-

(a) Delivery Agent Profile Data :- 1MB

(b) Order History & Earnings :- 9MB

(c) Location History & Tracking Data :- 4MB

Now, (a) Delivery Agent Profile Data Estimation :-

Storage required to store Delivery Agent Profile Data upto next year = $55M \times 1 \times 10^6$ Bytes

$= 55 \times 10^{12}$ Bytes

$= 55TB$

This data is stored in SQL dB and Delivery Agent Account Service is connected to it. This dB contains the delivery agent profile data for Authentication and Authorization purposes.

By Lossless decomposition, 30% of data can be saved.

So, Total Storage required in SQL DB = $0.7 \times 55TB$

$= 38.5TB$

(b.)

Order History & Earning Data storage Estimation

$= 55M \times 9 \times 10^6$ Bytes

$= 495 \times 10^{12}$ Bytes

$= 495TB$

Cassandra DB is used here which provide LZ4 compression algorithm by which 50% compression can be done easily without any data loss.

So, Data Storage Required = $0.5 \times 495TB$

$= 247.5TB$

(C) Location History & Tracking Storage Estimation (4MB)

(i) Neo4j :- 1MB (for recent movement, edges, nodes) —50%. Compression→ 0.5MB

(ii) Cassandra :- 3MB (Time-series Location History) —75% Compression→ 0.75MB

After Compression,

Storage Required = $(0.75+0.5) \times 55 \times 10^6 \times 10^6$ Bytes

$= 1.25 \times 55 \times 10^{12}$ Bytes

$= 68.75TB$

On Cassandra, LZ4 Compression is applied and in Neo4J Index-free Adjacency based compression is applied.

(c) For Restaurants :-

Let, Storage required to store for each restaurant
= 10MB

10MB Storage Breakdown

(a) Restaurant Profile Data :- 2MB (Name, Address, Contact, Images)
(b) Menu & Pricing :- 3MB (Food Items, Ingredients, Variants)
(c) Order History & Analytics :- 1MB → (Past Orders, Popular Dishes)
(d) Reviews & Ratings :- 1MB → (User Feedback, Average Rating)
(e) Operational Data :- 2MB → (Delivery Timings, Availability, Offers)

Total Restaurants (Assumed) = 100,0000
Next Year New Restaurants Added = 10% of 1000000
= 100000

Orders per Restaurant load = $\frac{0.1 \times 10^9}{10^5}$

= 1.2×10^2
= 100 (All restaurants assumed to be active)

In practicality, all restaurants, are not available.
Let 20% restaurants are unavailable
Then, Orders per Restaurant Load = 125
125 Orders are placed by each Restaurant per day

Restaurant Profile Data Estimation :-

Restaurant Profile Storage = $2 \times 10^6 \times 10^6$
= 2×10^{12} Bytes
= 2TB

The data is stored in SQL dB and Restaurant data storge for profile can be saved by lossless decomposition i.e 30% of data can be stored.

So, Total Storage Required in SQL dB = $0.7 \times 2TB$
= 1.4TB

For Menu & Pricing :-

Storage Required for Menu & Pricing $= 10^6 \times 3 \times 10^6$
$= 3 \times 10^{12}$ Bytes
$= 3$ TB

By Zstandard Compression Algorithm, we can save 40% of data storage.
Database Used :- MongoDB
So, Complete storage Required for Menu & Pricing after compression $= 0.6 \times 3$ TB
$= 1.8$ TB

For Orders History & Analytics :-
Storage Required for Order History & Analytics
$= 1 \times 10^6 \times 10^6$
$= 10^{12}$ Bytes
$= 1$ TB

By Parquet + Zstd Compression Algorithm 30% data can be saved.
So, storage Required after compression $= 0.7 \times 1$ TB
$= 0.7$ TB

Database Used :- Cassandra (Columnar DB)

For Reviews & Ratings :-
Storage required for Reviews & Ratings :-
$1 \times 10^6 \times 10^6$ Bytes
$= 10^{12}$ Bytes
$= 1$ TB

Database that can be used here :- MongoDB
Algorithm that can be apped to compress data :- Zstd
Saves 40% of data
So, storage required after compression = 60% of 1 TB
$= 0.6$ TB

For Operational Data (Delivery Timings, Availability, Offers) :-
Storage Required for Operational Data :- $2 \times 10^6 \times 10^6$
$= 2 \times 10^{12}$ Bytes
$= 2$ TB
Database that can be used :- Redis / PostgreSQL after 50% compression Algorithm, Data Storage Required = 1 TB

Total Restaurant data after compression to be stored upto next year $= (2.1 \times 0.7 + 3.3 \times 0.6 + 1.1 \times 0.7 + 1.1 \times 0.6 + 1.1 \times 2 \times 0.5)$ TB

$= (1.47 + 1.98 + 0.77 + 0.66 + 1.1)$ TB

$= 5.98$ TB

≈ 6 TB

Hence, Overall 33.33% compression is applied in approx.

Before compression, Storage Required for Restaurant

$= (2 + 3 + 1 + 1 + 2)$ TB

$= 9$ TB

We saved 3 TB of storage Here.

Complete Storage Estimation :—

= Restaurant Storage + Delivery Agent Storage + User Storage

$= 6$ TB $+ 11.76$ PB $+ (68.75 + 247.5 + 38.5)$ TB

$= 11.76$ PB $+ (68.75 + 247.5 + 38.5 + 6)$ TB

$= 11.76$ PB $+ 360.75$ TB

$= 11.76$ PB $+ 0.36075$ PB

$= 12.12075$ PB

≈ 12.5 PB (Taken Extra Storage as Buffer)

Now, Taking Replication factor = 3 to make our system highly available

Storage Requirement $= 3 \times 12.5$ PB

$= 37.5$ PB

for caching layer Taking Replication factor as 2.

Cache Data for User $= 4.1$ PB

Cache Data for Restaurant $= 0.2 \times 6$ TB ≈ 1.2 TB

Cache Data for Delivery Agent $= 0.2 \times 354.75$ TB

$= 70.95$ TB

Total Cache Data $= 4.1 + 0.07095 + 0.0012$

$= 4.17215$ PB

≈ 4.5 PB (Extra Storage for Buffer)

By Replication factor to increase Availability

Total Cache Data $= 9$ PB

Overall storage Estimation :- Cache Data + Storage Data

:- 9PB + 37.5PB

46.5PB

Caching Data Estimation for Restaurant & Delivery Agent -

For Restaurant -

(a) Profile Metadata :- 200KB

Number of Restaurant = 10 lakh

Since all Restaurants are not famous, we get maximum orders from Top restaurants ie 10-20% in total. So, searches will also go for them at max.

But still, we took Top 50% Restaurant data in cache in order to maximize the performance.

Hence, Total data required for Profile Metadata in Cache for restaurant = $200KB \times 5 \text{ lakh}$

$= 2 \times 10^2 \times 10^3 \text{ Byte} \times 5 \times 10^5$

$= 10^3 \times 10^3 \times 10^5 \text{ Byte}$

$= 10^{11} \text{ Byte}$

$= 100 \times 10^9 \text{ Byte}$

$= 100 \text{ GB}$

(b) Operational Data :- 50KB

Total Data required for restaurant in cache for operational purpose = $\frac{100GB}{4}$

$= 25GB$

(c) Popular Orders & Dishes :- 500KB

Total data required for restaurant in cache for popular orders & Dishes :-

$\frac{5}{2} \times 100GB$

$= 250GB$

Total data for cache in restaurant = $(100 + 25 + 250) GB$

$= 375 GB$

Cache Data Estimation for Delivery Agent :-

Caching required for :-

(a) Profile Metadata (Active Agents Only) → 100KB per agent

(b) Live Location & Tracking :- (Last 10 min) 300KB per agent

(c) Order history (Last 5 orders) → 150KB per agent

Now, Active delivery agent in total are 50 million

Total data for caching for delivery Agent

$$= (100+300+150) \times 10^3 \text{ Byte} \times 50 \times 10^6 \times 10$$

$$= 550 \times 10^3 \times 50 \times 10^6$$

$$= 50 \times 550 \times 10^9 \text{ Byte}$$

$$= 27500 \times 10^9 \text{ Byte}$$

$$= 27.5 \text{ TB}$$

Now, seeing compression how we can apply to reduce space need in order to minimize the space requirent and no data is lost

since, Total data for caching for Restaurant & Delivery agent = 27.5 TB + 0.375 TB

= 27.875 TB

Now, Algorithm for lossless decomposition in order to reduce Data need we are using is 6 Snappy Algorithms because of reason :-

→ compression Ratio :- 20 - 30%

→ Ultra-fast decomposition (Best for caching)

→ Low CPU Overhead.

If 25% compression is assumed.

then Data for cache in restaurant & Delivery Agent = 0.75×27.875

$$= 20.90625 \text{ TB}$$

$$\approx 21 \text{ TB} = 0.021 \text{ TB}$$

Overall storage Estimation = 46.5PB + 0.071PB
= 46.571PB
≈ 47PB (Assumed)

(B) Traffic Estimation :-

(i) Daily Active User Estimation -

Already Estimation Done ie 0.1 Billion

Since, All users are not active at the same time Requests will be distributed over 24hours and also the Request Rate is not uniform it will be high at certain hours and very low at specific hours.

Based on Daily Active Users we estimate:-

→ API request per second (RPS)
→ Peak Load
→ Concurrent Users (How many Users are Active at the same time)
→ Average Load

Assumed 10% of the users give requests at the same time

Peak Concurrent User = 10% of DAU
= 0.1 × 0.1 Billion
= 0.01 Billion

Assumed Each User gives 10 requests per min

peak load = 10 × 0.01 Billion/min
= 0.1 Billion/min
= 0.1×10^9 /min
= 10^8 /min
= $\frac{10^8}{60}$ requests per second
= $\frac{10}{6} \times 10^6$ Requests per second
= $1.67K \times 10^3$ RPS
= 1670K RPS
or
1.67M RPS

Average Load = $\frac{0.1 \times 10^9}{24 \times 60 \times 60}$ RPS ⇒ Average Load = 12K RPS

→ Read-Heavy system approx 80-90%.

→ API load estimation :-

(a) Product search API :- 40% of Total Queries (~0.67M RPS)

(b) Recommendation API :- 30% of Total Queries (~0.5M RPS)

(c) Order Placement API :- 10% of Total Queries (~0.17 M RPS)

(d) User Profile & Misc APIs :- 20% (~0.33M RPS)

(2) Computer Power Estimation

→ BackEnd servers Estimation (Application layer)

Assumed 500 requests are handled by each server

For Peak Load

Required Number of servers = $\frac{1.67M}{500}$

Assumed 500 RPS is accepted by one server or handled by one server alone

$= \frac{1.67 \times 10^6}{500}$

$= 2 \times 1.67 \times 1000$

$= 3.34 \times 1000$

$= 3340$ servers.

→ Database scaling (Read & Write) Request Handling (Assumed Peak Load)

Read Requests ~ $0.9 \times 1.67M$

Due to read-heavy system, we need replica database.

Assumed 1 DB instance ~ 50K reads handle

Required Read Replicas servers = $\frac{0.9 \times 1.67M}{50K}$

Since, Required Read Replica Servers = 30.06
We assumed it tobe 31
Hence, Required Read Replica servers = 31

Now, Write Requests ~ $0.2 \times 1.67M$
for writes, strong consistency is needed
Hence, we use horizontal scaling
1 strong Write DB instance ~ 10k Writes (handle)

Required Write Instances = $\frac{0.2 \times 1.67 \times 10^6}{10 \times 10^3}$

$= 0.2 \times 1.67 \times 100$
$= 0.334 \times 100$
$= 33.4$ servers.

So, Required DB Write Instances $\approx$ 34 servers

→ Caching Layer (Redis / Memcached)

To reduce DB load, we serve 80% read requests by cache alone.

Cache hit ratio = 80%.
Total Cacheable requests $= 0.8 \times 1.67M \times 0.9$
$= 1.2024M$
$= 1.2024 \times 10^6$
$\approx 1.3M$ requests

Each Redis / Memcached Node can handle 100k RPS.
So, Total Cache Nodes required = $\frac{1.3M}{100k}$

$= \frac{1.3 \times 10^6}{100 \times 10^3}$

$= 1.3 \times 10$
$= 13$

Total 13 Redis Nodes are required.

(c) Throughput Calculation:-

Already computed in Compute Ponce Estimation.

(d) Bandwidth Estimation:-

Bandwidth Estimation is Network load basically

Assume Average Request size = 50KB

Total Bandwidth required = $1.67M \times 50KB/s$

$= 1.67 \times 10^6 \times 50 \times 10^3$ Bytes/s

$= 83.5$ GB/s

If used Compression Techniques, we can reduce it by 50% i.e approx 42 GB/s

CDN & Caching will reduce the load on main servers that can be 21 GB/s too.

Why this assumed:-

(a) Light API Requests (User profile, Menu fetch etc.) → 1 KB - 5KB

(b) High Load Requests (Full Order History, Large Menus, etc.) → 50 KB - 500KB

(c) Media Upload:- (Profile Pic, Restaurant Image, etc.) → 1 MB+.

(d) order Placement:- (Cart, Address, payment, etc.) → 5KB - 50KB

Step-4:- Component Identification:-

Services that comes in my mind are given below:-

- User Account service
- User Authorization service
- User Authentication service
- Load Balancer
- Distributed zookeeper service
- Request Handling service
- Location Manager service
- Search Page service
- Order History service
- Cart service
- Checkout Service
- Inventory Service

Order Status service

Payment service

Packing service

Notification service

Restaurant Manager service

Driver / Delivery Agent Management service

Recommendation service

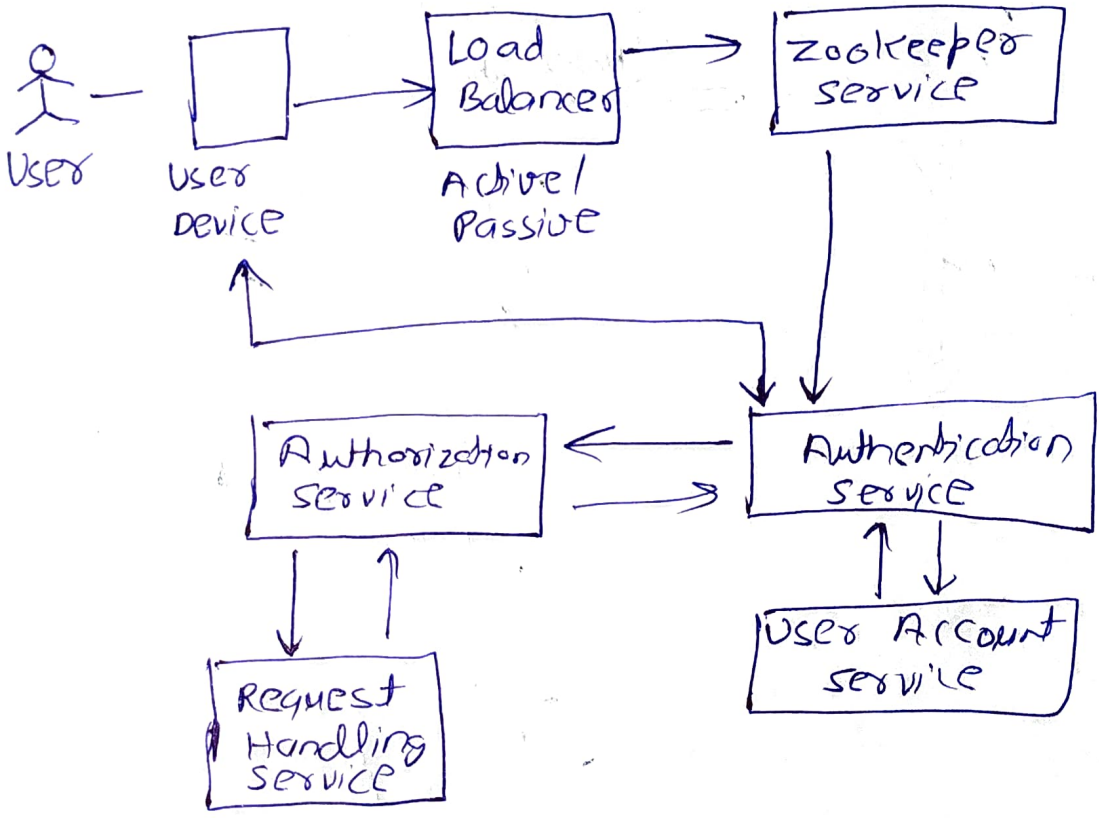
Analytics & Insight service

Confirmation Message service

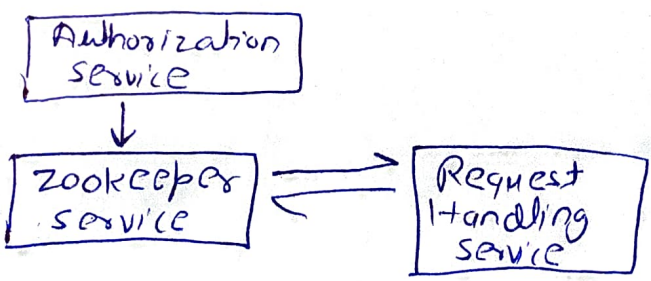
Review & Rating service

Customer Support service

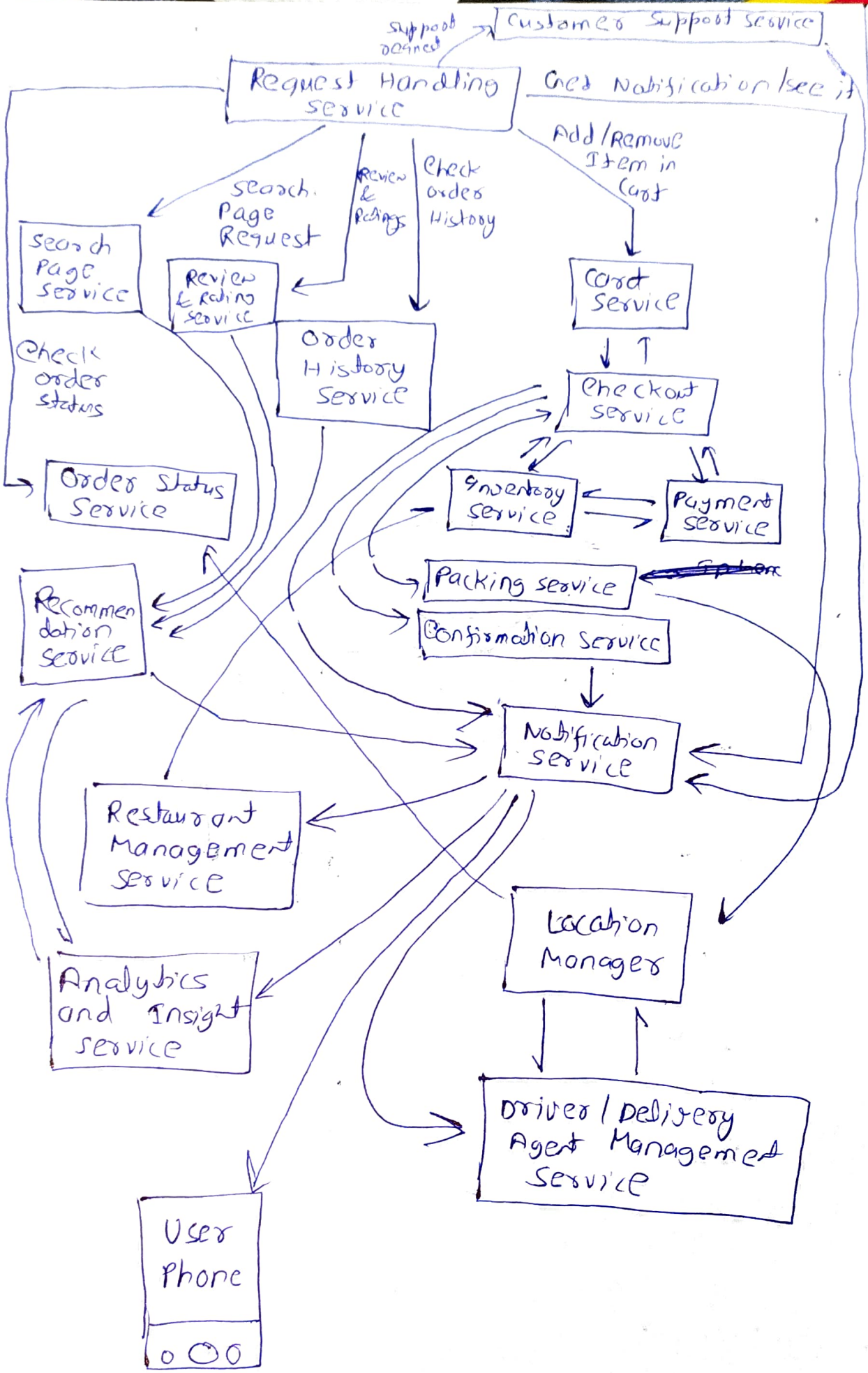
How service interact with each other :—



Now, Request Handling service working



Request Handling service is connected to different types of services


Support request
Customer Support service
Request Handling service
Get Notification/see it
Search Page Request
Review & Ratings
Check order History
Add/Remove Item in Cart
Search Page service
Review & Rating service
Order History service
Cart service
Check order status
Checkout service
Order Status Service
Inventory service
Payment service
Packing service
Confirmation service
Recommendation service
Notification service
Restaurant Management service
Location Manager
Analytics and Insight service
Driver / Delivery Agent Management service
User Phone

More Modular Design Thought

Notification service has high load. To reduce the load. We introduced asynchronous communication by introduction of Messaging Queue KAFKA.

All Notification will be pushed to KAFKA and Notification service will become consumer and send the message to the required service.

Where to use Zookeeper :-

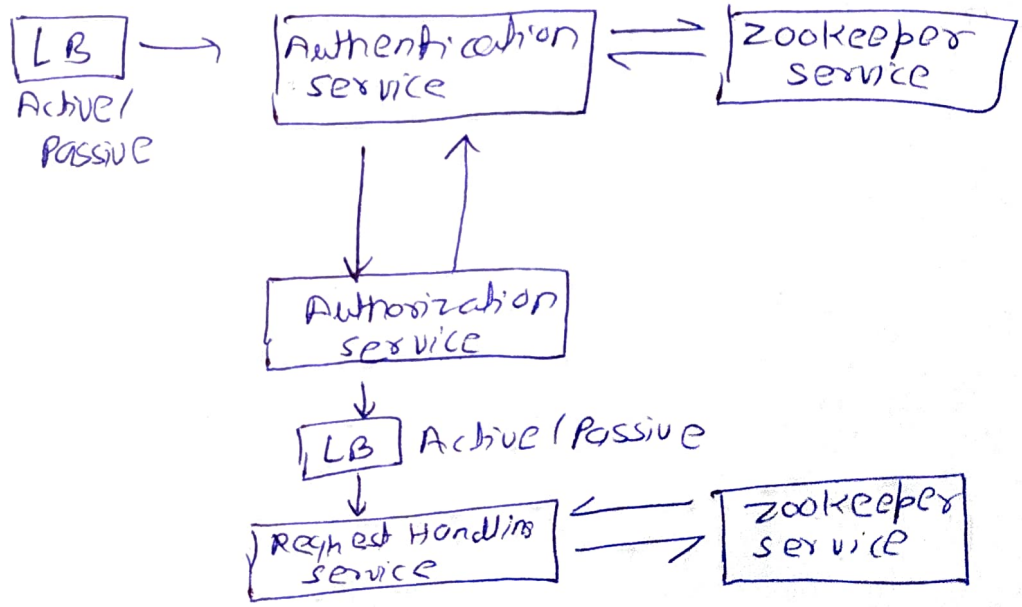
Analytics & Insights service ⇄ Zookeeper
Checkout service ⇄ Zookeeper

Sequence of Load Balancing Algorithm :-

Round Robin
↓
Least Connection
↓
Weighted Round Robin

↓ Request per second ↑se

Hardware load Balancers :-



We changed the approach that LB will be directly connected to zookeeper. Wherever Zookeeper is needed, we connect it over the backend service

Introduction to ID Generator service

ID generator service is needed to produce unique ID based on the content given to it. To produce unique id for users, transactions etc. since our application is read-heavy separate services are maintained for ID generation. → We use Master-slave Architecture

Stateless system are preferred:-

(a) Scalability:- Easily horizontally scalable.
(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

Strategies for Distributed Database

Master-Master Architecture for Distributed Database:-
(Preffered where significant writes are taking place)
(a) Payment service (b) Delivery agent Management service
(c) Request Handling service (d) User Account Service
(e) Authorization Service (f) Authentication Service
(g) Checkout service

Master-Slave Architecture for Distributed Database:-
(a) Location Managers Service (b) Search Page service
(c) Order History service (d) Cart Service
(e) Confirmation message service (f) Recommendation service
(g) Analytics and Insights Service (h) Notification Service
(i) Restaurant Management Service (j) Review & Ratings service (k) Wishlist service
(k) Packing service (l) Order Status Service

Combined Master-Master & Master-Slave Distributed DB:-
(i) Inventory Service (ii) Customer support Service

Noted Point:- All services are Distributed in nature. We have specified Hardware Load Balancing by [LB] explicitly. For service to service communication (distributed both) we always require LB.

Hence, we need to choose what to choose Software Load Balancing or Hardware Load Balancing.

If we have not mentioned LB b/w two services, By default it is assumed that Software Load Balancing is used.

Now After chosen software and Hardware Load Balancing, we decide what to choose either L4 or L7. This decision will directly depend on what communication protocol will be involved between two services.

Decision on Database:-

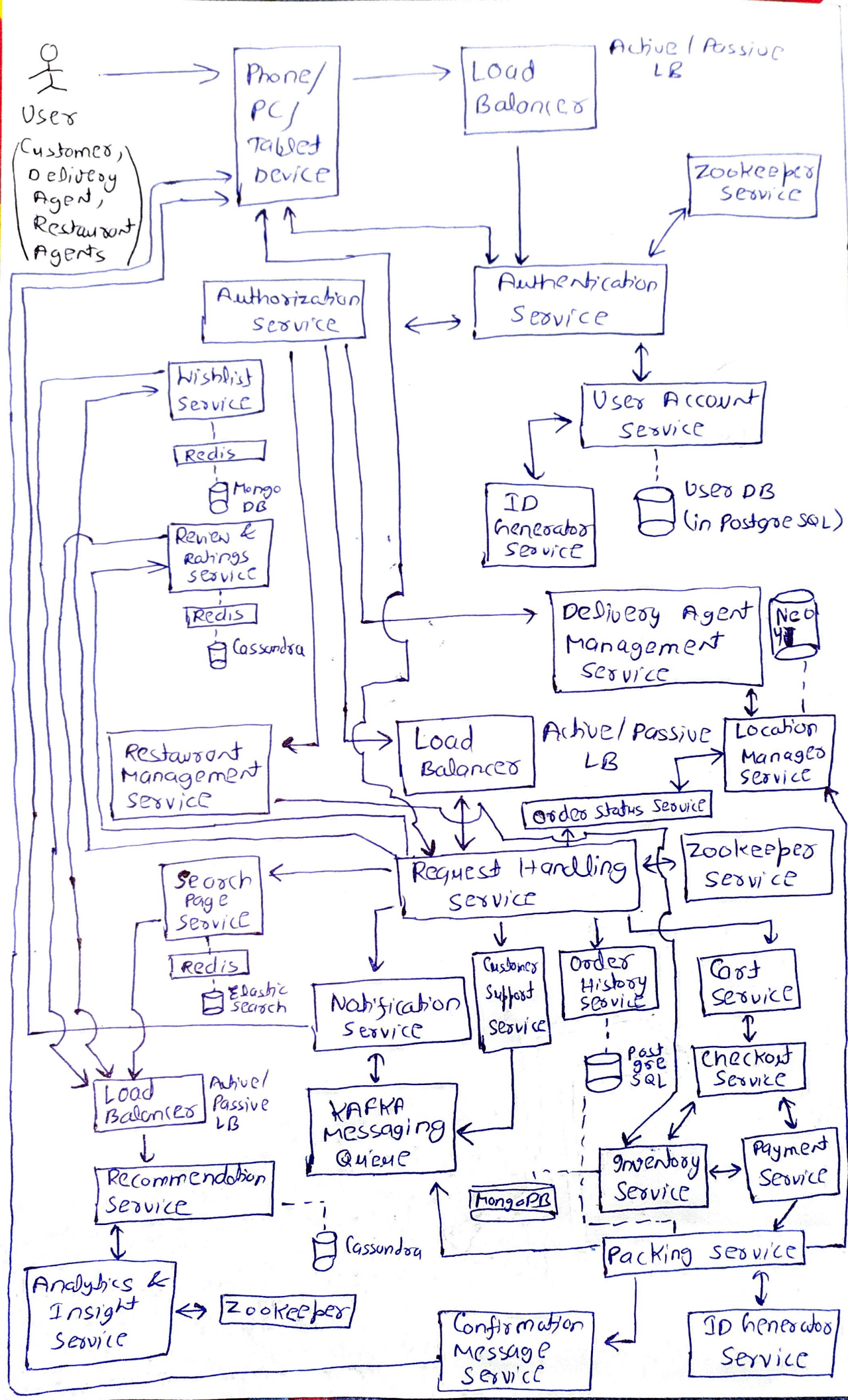
Sharding is used and after sharding, we use the range-based partitioning

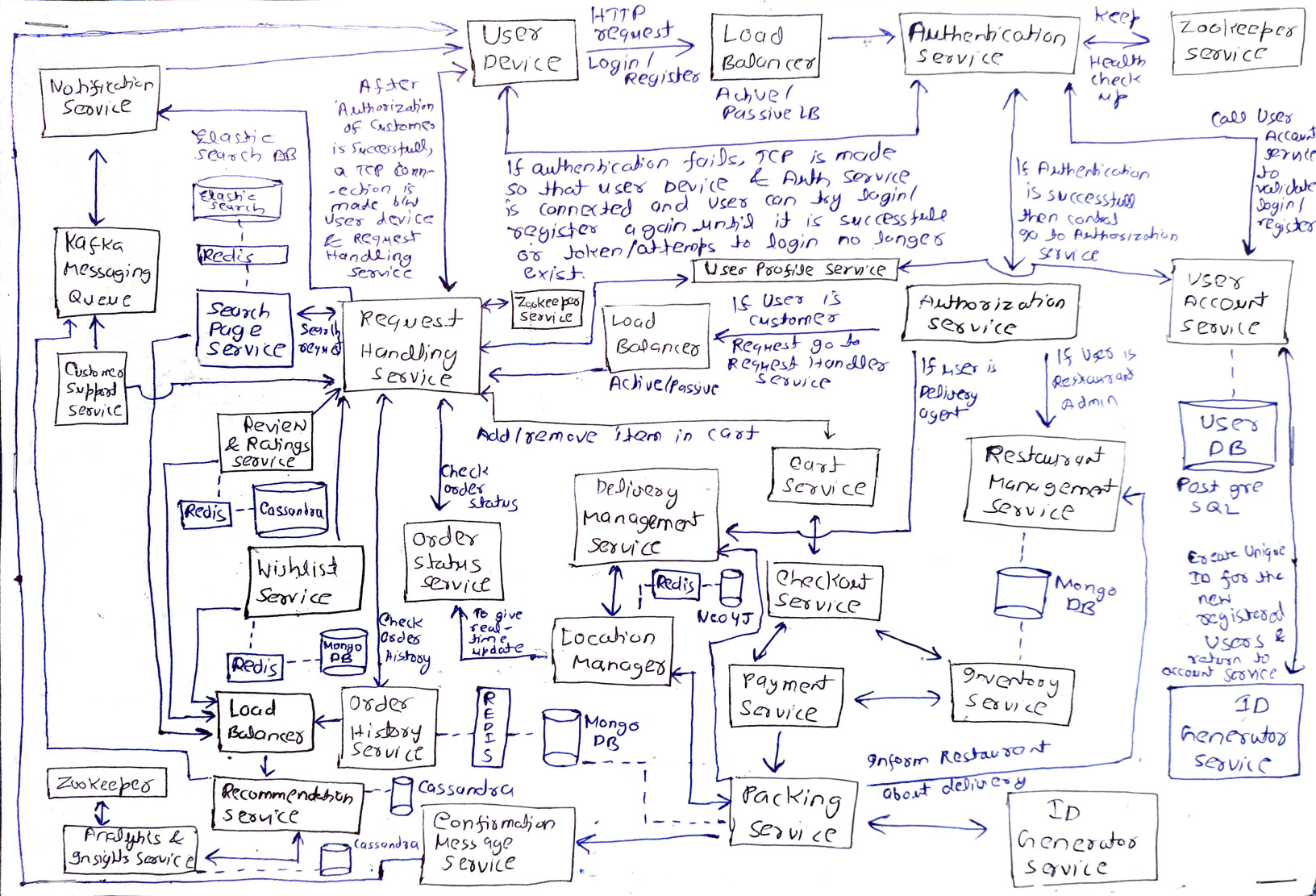
Database Selection:-

① Wishlist service (New Addition) → Mongo DB
② Recommendation service → Cassandra
③ Location Manager service → Neo4j
④ Account service → PostgreSQL
⑤ Checkout service → MongoDB
⑥ Inventory service → MongoDB
⑦ Search Page service → Elastic search

We also introduce cache (Redis) in b/w service and DB to increase speed of our system performance. Final Design is now proceeded as given below:-

Final Design :- (Step-5)


User
(Customer, Delivery Agent, Restaurant Agents)
Phone/ PC/ Tablet Device
Load Balancer
Active / Passive LB
Zookeeper Service
Authentication Service
Authorization Service
User Account Service
ID Generator Service
User DB (in Postgre SQL)
Wishlist Service
Redis
Mongo DB
Review & Ratings Service
Redis
Cassandra
Delivery Agent Management Service
Neo 4j
Restaurant Management Service
Load Balancer
Active/ Passive LB
Location Manager Service
Order Status Service
Request Handling Service
Zookeeper Service
Search Page Service
Redis
Elastic Search
Notification Service
Customer Support Service
Order History Service
Cart Service
Checkout Service
Postgre SQL
Load Balancer
Active/ Passive LB
KAFKA Messaging Queue
Inventory Service
Payment Service
Recommendation Service
MongoDB
Cassandra
Packing Service
Analytics & Insight Service
Zookeeper
Confirmation Message Service
ID Generator Service


User Device
HTTP request
Login / Register
Load Balancer
Active/Passive LB
Authentication Service
Keep Health check up
Zookeeper service
Notification Service
Elastic search DB
Elastic search
Redis
Kafka Messaging Queue
After Authorization of Customer is successful, a TCP connection is made b/w User device & Request Handling service
If authentication fails, TCP is made so that user device & Auth service is connected and user can try login/register again until it is successfully or token/attempts to login no longer exist.
Call User Account service to validate login/register
If Authentication is successful then control go to Authorization service
User Profile service
Search Page Service
Search request
Request Handling Service
Zookeeper service
Load Balancer
Active/Passive
If User is customer
Request go to Request Handler service
Authorization service
User Account service
Customer Support service
If User is Delivery agent
If User is Restaurant Admin
Add/remove item in cart
Review & Ratings service
Redis
Cassandra
Check order status
Order Status service
Cart Service
Delivery Management Service
Restaurant Management Service
User DB
Postgre SQL
Wishlist Service
Redis
Mongo DB
Check Orders History
To give real-time update
Location Manager
Redis
Neo4J
Checkout Service
Mongo DB
Create Unique ID for the new registered Users & return to account service
Load Balancer
Order History Service
REDIS
Mongo DB
Payment Service
Inventory Service
ID Generator service
Zookeeper
Recommendation service
Cassandra
Inform Restaurant about delivery
Analytics & Insights Service
Cassandra
Confirmation Message Service
Packing Service
ID Generator service

Communication Protocol and L4/L7 Load Balancing selection

Service 1	Service 2	Communication used	L4/L7 LB
User Device	Load Balancer	HTTP	L7
Load Balancer	Authentication Service	HTTP	L7
Authentication Service	Zookeeper service	TCP	L4
Authentication Service	User Account Service	TCP	L4
User Account service	ID Generator Service	TCP	L4
Authentication Service	Authorization Service	HTTP	L7
Authorization Service	Load Balancer	HTTP	L7
Load Balancer	Request Handling service	HTTP	L7
Authorization Service	Delivery Management service	HTTP	L7
Authorization Service	Restaurant Management service	HTTP	L7
User Device	DMS / RMS	Socket.io	L7
User Device	Authentication Service	TCP	
User Device	Request Handling Service	Socket.io	L7
Confirmation Message service	User Device	UDP	L4
Notification service	User Device	UDP	L4
service	KAFKA	UDP	L4
Customer support service	KAFKA	UDP	L4
Request Handling service (RHS)	Wishlist Service	HTTP	L7
RHS	Review & Ratings Service	HTTP	L7

Service 1	Service 2	Communication Protocol	L4 / L7 LB
RHS	zookeeper	TCP	L4
Analytics & Insight service	zookeeper	TCP	L4
RHS	Order History service	HTTP	L7
Wishlist, Search, Review & Rating Service	Load Balancer	UDP	L4
Load Balancer	Recommendation service	UDP	L4
Recommendation service	Analytics & Insight service	UDP	L4
RHS	Cart service	HTTP	L7
Cart service	Checkout service	HTTP	L7
Checkout service	payment service	TCP	L4
Payment service	inventory service	TCP	L4
Payment service	Packing service	UDP	L4
Packing service	Restaurant Management service	UDP	L4
Packing service	ID generator service	TCP	L4
KAFKA	Notification Service	UDP	L4
order status service	Location Manager	socket.io	L7
Notification service	Request Handling Service	HTTP	L7
Packing Service	Confirmation Message Service	UDP	L4
Location manager	Packing service	UDP	L4
Database, Cache Layer	Other services	TCP / UDP (UDP is preferred where we need speed and data loss is acceptable)	L4

How System interact with each other?

Key features of final design not explained in detail

① All services are distributed in nature i.e all services follow the distri-buted Architecture.

② Those services which are getting heavy requests are maintained by either Hardware Load Balancing or the combination of Zookeeper + Software LB

③ Note:- After Authorization user device maintains a direct web socket.io connection with a request handling service, restaurant management service and delivery management service for customer, restaurant admin and delivery agent respectively.

④ Request Handling service respond to different types of request and for every different request, different services, are respectively called.

⑤ Load Balancer shown explicitly are Hardware LB while b/w every distributed service LB is required so if LB not shown means by default we are using software Load Balancer.

⑥ How Cart Service works:-

(a) Cart service make a call to checkout service.

(b) Now, checkout service will call to inventory service to see if item available or not, inventory service respond back to checkout service and if item available keep a temporary freeze on the item.

(c) once that happened, checkout service call to payment service & payment service call to inventory service to deduce the item quantity and call to packing service. Packing service call to location manager to give location of user's home (customer's delivery place), alert both restaurant & Delivery management service Also it call to confirmation Message service that specify / alert user that order is finalized / booked.

But Packing service all these calls will be asynchronous in nature and all these asynchronous call are called after

unique ID is returned by ID generator service.

(d) Packing service will not begun it's call to other service until ID generator service Response is returned.

(7) Active/Passive LB is used in order to provide high availability to our system

(8) How Recommendation service works

(a) Review & Ratings service, Wishlist service, Order History service and search page service call tells about which restaurant is which dish is more saled out. To increase the chance of buying a dish per visit of app, the data of all these service plays a crucial role. Based on these service responses, recommendations are given so that buy rate ↑se.

(b) since, 4 services are calling to Recommendation service, we used HardWare Load Balancer to distribute requests more evenly.

(c) Now, Recommendation service will send these results to Analytics & Insights service and based on output of AEI service, Recommendation service will give recommendation to user in form of notifications.

(d) The rate of request of notification can go in any order and order not matters that's why we used KAFKA messaging queue, also the rate of requests will be also very high. that's why asynchronous communication is introduced here.

(9) Restaurant can update their menu which should be notified to the inventory service that's. Restaurant Management & Inventory service both are connected to some DB.

This maintains consistency in the database of Inventory Service.

(10) Order Status service is connected to Location manager to give real-time update to user wherere it's product / order arrived or to give perfect ETA (Early Time Arrival Estimation)

(11) Packing Service also updates data of Order History service since new order is placed

(12) Location Manager is connected to Delivery Management service since Delivery Management Service updates the Location to Location Manager

Now, We can briefly define how Delivery Management Service and Restaurant Management service looks like:-

(a) Delivery Management Service

(i) Delivery History Service

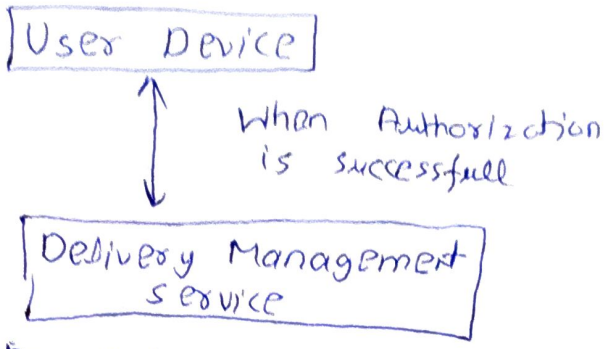
(ii) Customer Detail Service

(iii) Delivery Panel Service

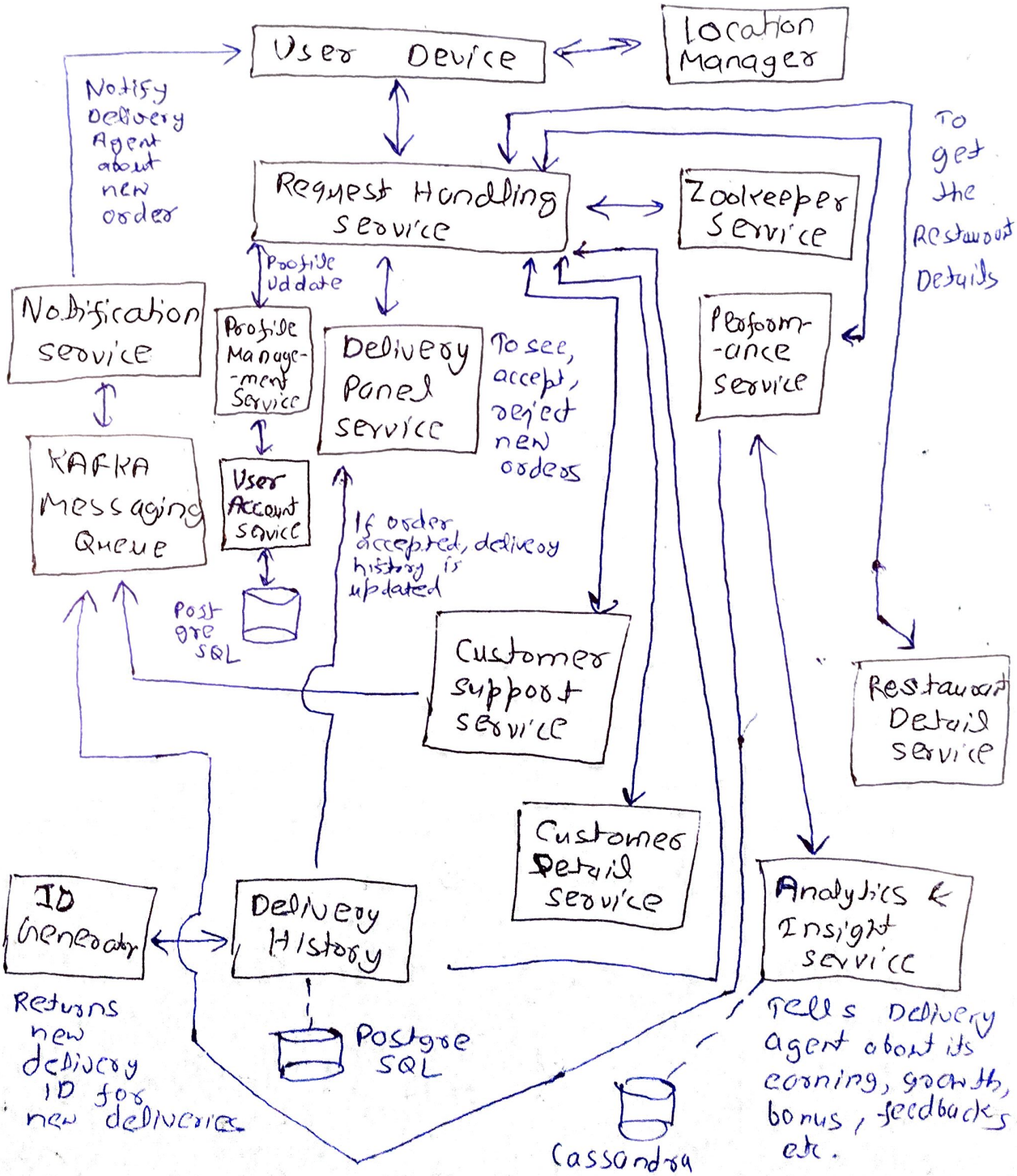
(iv) Performance Service

(v) Restaurant Detail service

Delivery Management service looks as given below:-



Breakdown of DMS:-



Breakdown of Restaurant Management Service