**Uber**

Functional Requirements:

1. See the cabs around you.
2. ETA & approx. Price of how much it would cost you.
3. Customer should be able to Book a Cab
4. Live Location Tracking of the booked cab.

Non-Functional Requirements:

1. Global: Must be available in all the locations. Servers need to be present in a lot of places
2. Low Latency
3. High Availability: Certain components need to be highly available.
4. High Consistency: Certain components need to be highly consistent.

Numbers:

1. Uber has 100 million active users monthly. It has 14 Million rides/day.

Problem at hand

1. When you have customer’s location, find the nearest few drivers, and then choose the best driver of them who is best suited for the customer and driver.
   1. **Map Service**:
      1. Divide the city into a segments
      2. Given the latitude & longitude of a customer and Latitude and Longitude of some cabs, find the segment to which customer belongs to & return a list of drivers near him.
      3. This service also powers the ETA, Distance, Route functionality.
      4. This service should be able to merge and divide the segments further so that it can handle less traffic of cabs in one segment as well as more traffic of cabs in one segment.

**User Flow:**

Diagram

Description automatically generated

1. User App talks to a Load Balancer which talks to the User Service.
2. User Service: This is the repository of all the user information + it will talk with other services which will give the user the information which it wants. Ex: User Profile is powered by User Service. Also, if a user wants to see ‘My trips’, then user service will call the Trip Service and send it back to the user. It stores all the details in a My SQL DB and uses Redis as Cache. If user wants to see his profile, it will call REDIS, and if the data is not there, it will call the DB and store the result in Cache and return to the user.
3. Cab Request Service: It is connected through a web socket connection with the user app. It places a few cabs which are around the user with their location. It places a request to the CAB FINDER service and return the response.

**Driver Flow:**

A picture containing text, whiteboard

Description automatically generated

1. Driver App calls the Driver Service like User Service.
2. Driver Service: This is like the UserService. If a driver wants to see the payment information, driver service will call the payment service internally which can fetch the payment details from the payment service. It also sits on top of the Mysql and Redis.
3. Driver app also talks with the Location Service through a series of servers maintaining a web socket connection. When a driver moves through a city, every 5 secs, their location is sent to the location service, which will query the map service to find out which segment the driver belongs to.
4. With the help of the customer segment and the segments of a few drivers in some segments, the best suited driver for a customer is calculated through the User’s “Cab Finder” and a few other services.

**Why is web socket needed?**

1. For each driver, we must have a web socket connection to the backend.
2. Driver needs to send his location to the backend server continuously (every 5 secs).
3. Servers might want to talk to the driver when a booking is assigned, so we can use the same Web Socket Connection with the booking details.
4. Customer must see the few cabs which are near to his location continuously, so we need a web socket connection here as well.

**System Design:**

Diagram

Description automatically generatedDiagram

Description automatically generated

1. **Web Socket Manager:** In a real word example, there will hundreds of web sockets for each of the drivers. If we need to send a Booking confirmation to a driver, then we will have to search it in thousands of web sockets and here this web socket manager will help. It is a distributed service which manages the fact that which Web socket is connected to which driver. When a new driver comes online and gets connected with a web socket, it also informs the Web Socket Manager about it.
2. **WSM -> REDIS:** It stores all the information in REDIS database in memory + disk.
   1. It will store the mapping between Web Socket Connection Handlers (host) and Drivers
   2. It will also store drivers -> host/Web Socket Handler mapping.
3. **Location Service:** Every 5 secs, the driver app would send the location through the web socket.
   1. It stores the LIVE LOCATION detail of the driver in Cassandra. Why Cassandra? => Millions of drivers are sending updates every 5 secs so a lot of updates are happening, and Cassandra can scale to that level.
   2. While the trip happens, we need to know the exact route which the driver followed since it may be required for auditing, calculating bill, etc. For this, we need to store all the location pings that the driver app was sending.
   3. Location Service talks with **Map Service** to give us the ETA, Distance, Route. It stores the driver segment into REDIS database, and this will tell us that the segment X has how many drivers right now. This update happens only when segment changes.
4. **Trip Service:** Source of Truth for all the trip related information. It sits on top of Cassandra and MySQL Database.
   1. MySQL is used for all the trip which are LIVE or which are scheduled and about to happen.
   2. As soon as a trip is completed, we move the trip to Cassandra Database.
   3. Why don’t we store everything in MySQL? Overtime there will be a lot of data and we would just want to have read queries so we can store it in Cassandra where the reads are good enough.
   4. Why are we storing it in MySQL: It will have a lot of information about drivers, customers, start time, distance, events, payments. There will be a lot of tables and for each event that happens in the trip, we may want to update a lot of tables and there it is good to have transactional property of MySQL.
   5. This Trip Service will expose all the APIs like getTripByID(), allTripsOfAUser(), allTripsofaDriver() which will collate the data from MySQL and Cassandra and return.
   6. **Trip Archiver:** Service which once in every 12hrs pulls data from MySQL and pushes into Cassandra.

**Customer Flow:**

1. Customer calls the Cab Request Service through a Web Socket Connection with the arguments of source latitude, longitude, destination latitude, longitude, Type of Cab, etc.
2. Cab Request Service would call the Cab Finder Service with all the relevant arguments. This Service would finally return me 1 driver who will be assigned this trip. It will return TripId, DriverId, CarDetails, etc. This will be returned to the customer.
3. CabFinder Service will also put a message in Kafka queue whether it was able to find a driver for that source latitude, longitude. This can be used for Analytics.
4. **Cab Finder:** 
   1. It will send the source latitude, long to the Location Service and it will tell it the segment in which the customer is in currently.
   2. Along with segment, it also responds with a list of drivers which are near to this segment. How it does that? => It queries the surrounding segments + the customer segment and then gets a list of all the drivers from this point. Then we need to find the closest 10 drivers from all those drivers.
   3. Cab finder has a list of some drivers who can do the trip but it decides which one out of them on the basis of the different modes, Eg: Premium Customer then best driver. Average customer, then broadcast and whichever driver accepts first is assigned, etc. This is done by the Driver Priority Engine.
   4. **Driver Priority Engine:** Given a list of drivers and a mode, it prioritizes them and returns it to the cab finder service. Cab finder service will not find one driver from the list.
   5. Then given the driver, it queries Web Socket Manager asking for the host/Web Socket which was talking to this driver. If the driver is not connected, we will look for other drivers in the queue.
   6. It will send a notification to the driver about the Trip, Customer details.
   7. It will send a notification to the customer about the Trip, Driver details.
   8. This service will also update the TRIP SERVICE with all the customer, driver, details.

**Analytics / Kafka Events:**

A picture containing text, whiteboard

Description automatically generatedDiagram

Description automatically generated

1. Kafka gets a lot of events like Trip Update Event, Location Update Event, No driver Find Events, Trip Completed, etc.
2. **Payment Service:**
   1. This service would sit on top of Kafka and have a Kafka Consumer which would listen for Trip Completion Events.
   2. Once it reads an event, it will put all the data into the payment mySQL database saying this driver did this particular trip with this distance, this time, customer Id.
   3. This can talk with a payment gateway for the payment.
   4. This service would also expose APIs like getAllPaymentInfoForDriver()
3. Spark Streaming:
   1. We can create a heat map of driverNotFound events and ask drivers to move to those locations so that they can serve the customers in that area.
4. Hadoop Cluster: We can dump all the events into a Hadoop Cluster.
   1. We can run a lot of Hadoop / Spark jobs which can do:
   2. Customer Classification: Premium/Non-Premium Customer/Drivers
   3. Driver Priority Engine can be powered by ML/AI from here.
   4. Fraud Engine: If a driver-customer are always together, then it may be fraud.
   5. Map Service: The traffic information can be calc by our drivers at a certain location. Road Conditions Can also be taken into account. It can be used for better ETA prediction.