

January 2024

SMART CAB ALLOCATION SYSTEM

FOR EFFICIENT TRIP PLANNING

Prepared by:
ishaan Mittal
210003039

Introduction

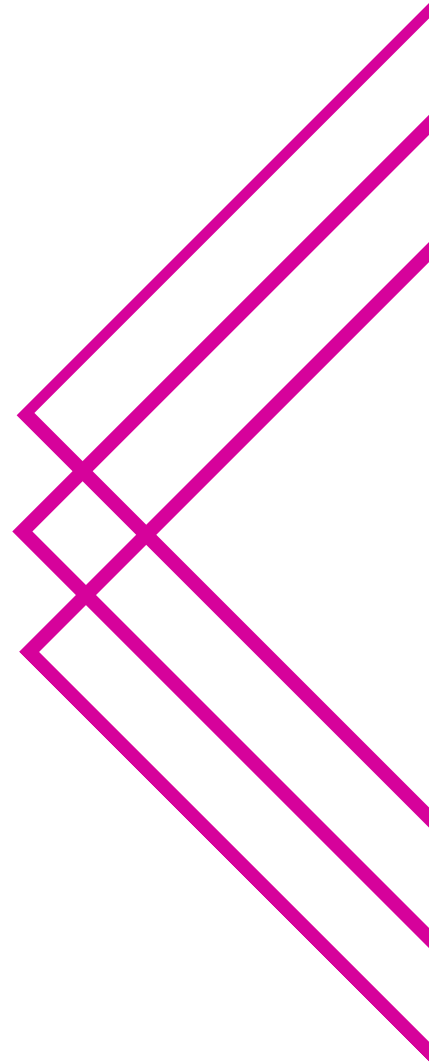
The Smart Cab Allocation System represents an innovative and efficient solution for optimizing the process of allocating cabs for trips, focusing on enhancing overall travel efficiency and user experience. In today's fast-paced world, the demand for reliable and streamlined transportation services has never been higher. This system sets new standards with a proximity-based algorithm for administrators, minimising travel distance and offering employees a user-friendly interface with real-time suggestions for nearby cabs. This design integrates real-time location data into it.

Roadmap

The implementation roadmap for the Smart Cab Allocation System involves a sequential progression through key milestones.

1. Establishing a secure authentication system, employing industry-standard protocols like OAuth 2.0 or JWT, and integrating multi-factor authentication for heightened security.
2. database schema design will be made, and the selection of suitable databases to meet system requirements.
3. Careful selection of an Object-Oriented Programming Language (OOPS) to serve as the backbone of the system, with emphasis on modular, maintainable code adhering to OOPS principles.
4. The integration of real-time location access using the Google Maps API enhances system accuracy and responsiveness.
5. Finally, the implementation journey concludes with the Dockerization of the application for consistent deployment and the integration of Kubernetes for efficient orchestration, ensuring scalability and resilience in the Smart Cab Allocation System.

We'll be taking care of error and exception handling at every step in the roadmap.



Project Overview

The Smart Cab Allocation System caters to three key user roles: Users, Drivers, and Administrators, each given distinct functionalities tailored to their specific needs.

Users:

- **Cab Booking:** Users can effortlessly browse available cabs, retrieve information on their locations, and book a cab for their desired destination.
- **Trip History:** Users can access a comprehensive trip history log, offering insights into previous bookings, routes taken, and fare details for improved record-keeping.

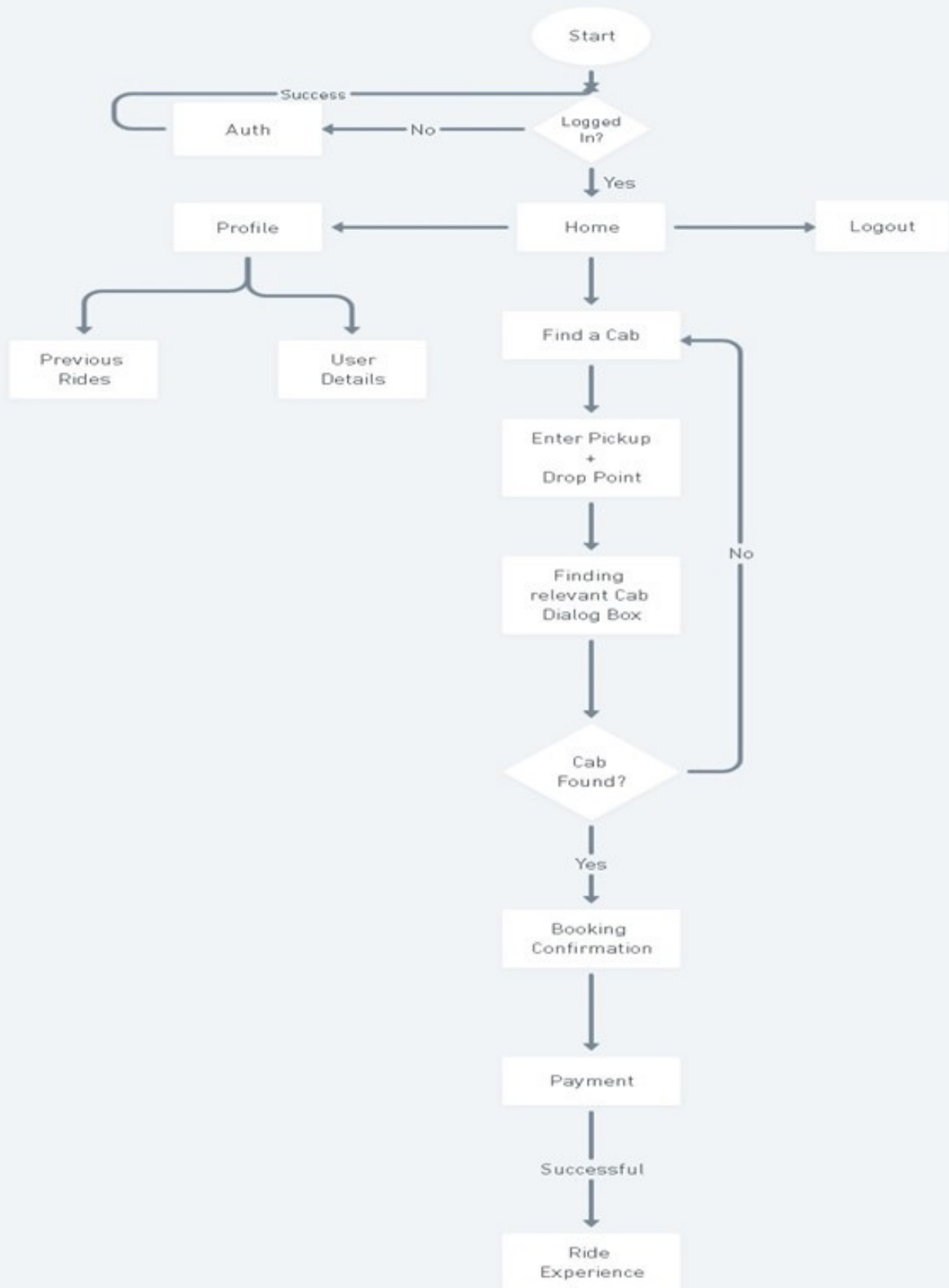
Drivers:

- **Trip Acceptance and Rejection:** Drivers have the autonomy to accept or reject trip requests based on their availability and location, ensuring flexibility in their schedules.
- **Real-Time Navigation:** The system provides drivers with real-time navigation assistance, optimising routes and estimated arrival times for efficient trip completion.
- **Performance Analytics:** Drivers can access a dashboard showcasing trip history and performance analytics, empowering them with insights for continuous improvement.

Administrators:

- **Cab Allocation Optimization:** Cab allocation algorithms are implemented to ensure efficient and strategic allocation of cabs based on their proximity and time needed to reach the user.
- **User and Driver Management:** Administrators have the authority to manage user and driver profiles, including account creation or removal, ensuring a secure user base.

User Flow Chart



Authentication

The libraries and mechanisms that we will be using for implementing a secure authentication.

Bcrypt: Secure Password Hashing

- Bcrypt is a robust library employed for secure password hashing, incorporating salting to fortify the hashing process. Salting ensures that even if users share identical passwords, the resultant hashed representations are unique, significantly enhancing password security.

JWT: JSON Web Token for Secure IdentityRepresentation

- JSON Web Token (JWT), serves as a secure and cryptographically signed representation of the user's identity. JWTs provide a standardised and secure method for representing user identities within the authentication process. By digitally signing the token, the integrity of the user's identity claims is assured, preventing tampering or unauthorised modifications. This mechanism enhances the overall security and reliability of user sessions.

OAuth 2.0 (GoogleLogin): Secure User Authentication via Google Services

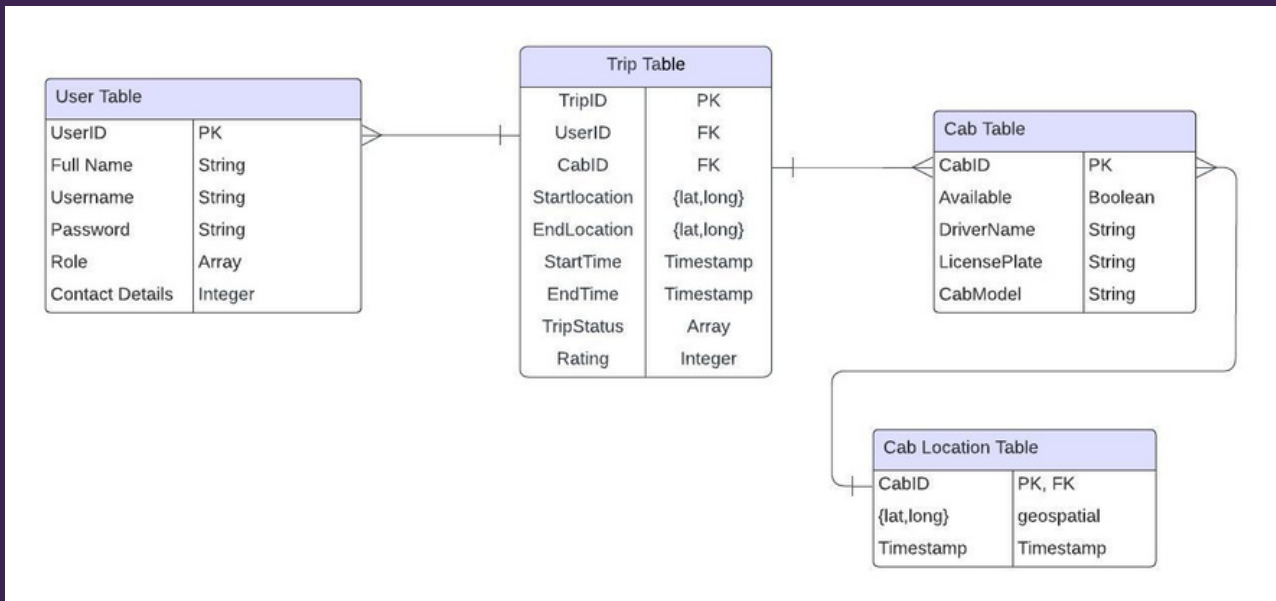
- OAuth 2.0 is an authorisation framework that enables secure user authentication through Google's services. This mechanism grants the application limited access to user accounts on its HTTP service by delegating the authentication process to Google's hosting service.
- Leveraging OAuth 2.0 for Google Login ensures a secure and standardised approach to user authentication. By relying on Google's authentication service, the application minimises handling of sensitive user credentials, enhancing overall security. This framework also simplifies user access, as users can utilise their existing Google credentials for authentication.

We will be using all three of them together to make a secure authentication system for our application.

Authentication Flow Chart



Database



We will be using a combination of MongoDB and Redis for the database architecture of our application. MongoDB will host the User, Trip, and Cab tables, offering flexibility and efficient document-oriented storage. Redis will manage the Cab Location table, leveraging its support for geospatial data, spatial queries for location data and low-latency updates, important for real-time location tracking.

- User Table (MongoDB): Stores user-related information.
- Cab Table (MongoDB): Manages cab-related data such as availability, driver details, and model.
- Cab Location Table (Redis): Efficiently stores real-time geospatial location data for cabs and utilizes Redis's geospatial indexes for spatial queries.
- Trip Table (MongoDB): Records trip-related information, including user and cab details.

This combination of MongoDB and Redis aims to provide a robust and scalable foundation for the Smart Cab Allocation System, addressing the specific needs of structured data storage and real-time geospatial tracking.

Real-time location data integration

Real-time location data integration is a crucial component of any modern ride-sharing platform. It allows the system to track cabs continuously, ensuring that the information used for making decisions about cab allocation is as accurate and timely as possible.

Cab-Side Implementation:

- Each cab is equipped with a GPS device or a smartphone app capable of determining its current location.
- The device/app is configured to push the location data to the server at predefined intervals, say every 10 seconds. This interval can be dynamic based on the state of the cab (e.g., more frequent when it is engaged in a trip).
- The cab's device/app authenticates with the server via secure protocols to establish a session for the subscription.


Server-Side Implementation:

- The server maintains a persistent connection or listens for incoming location updates from each cab.
- It authenticates the source of the data to prevent spoofing or unauthorised access.
- Upon receiving an update, the server processes it immediately or places it into a queue for batch processing, depending on the system's design for handling real-time data.

Data Handling:

- The server uses Redis which has geospatial capabilities, to update the cab's location. This index allows for efficient spatial querying of location data.

Location Update Algorithm:

- When a location update is received, the algorithm checks if the reported location is significantly different from the last known location to avoid unnecessary updates.
 - If the location has changed beyond a certain threshold, the cab's entry in the geospatial index is updated.
 - The dynamic map is also updated to reflect the new location and potentially the cab's availability status.
- 

Fault Tolerance and Reliability:

- The server is designed to handle missed updates gracefully. If a cab does not send an update for a certain period, its status may be set to "unknown," and it won't be considered for new ride allocations until it re-establishes communication.
- Redundancy is built into the communication system to handle cases where the primary update channel fails.

Scalability:

- The server's infrastructure is designed to scale horizontally with the use of a distributed database(MongoDB) and containerization(Docker and Kubernetes)
- Load balancers like gunicorn distribute the incoming data across multiple servers.
- The geospatial index and dynamic map can be sharded or replicated across multiple servers to handle read/write loads efficiently.

Security Considerations:

- All communication between the cabs and the server is encrypted using TLS/SSL to protect the privacy and integrity of the location data.
- Access to the location update API is restricted to authenticated devices/apps to prevent unauthorised data injection.

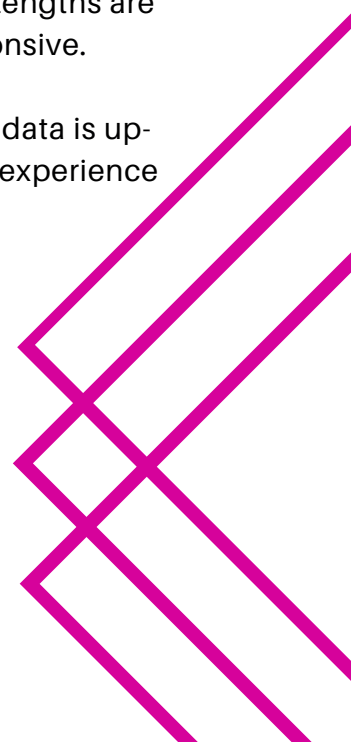
Optimisations:

- The system may implement predictive algorithms that use historical data to anticipate a cab's future location, reducing the dependency on constant real-time updates.
- Data aggregation techniques can summarise location information when precise details are not necessary, reducing the amount of data stored and processed.

System Monitoring and Alerts:

- The platform includes monitoring tools to track the flow of location data, alerting system administrators if data streams fall below certain thresholds, indicating potential issues with the cab-side devices or network connectivity.
- System metrics such as update latencies, processing times, and queue lengths are monitored to ensure the real-time system remains performant and responsive.

Through this detailed approach, the platform ensures that the cab location data is up-to-date, allowing for efficient and accurate cab allocation and optimal user experience for both drivers and passengers.



Employee's Cab Search Optimization

The objective of the Employee's Cab Search Optimization is to create a system that enhances the user experience for employees who are looking for cabs. This system focuses on providing employees with information on cabs that are currently in use but are near their location and likely to become available shortly.

- **Real-Time Cab Status Tracking:** The system maintains a real-time status of all cabs, marking whether they are currently engaged in a trip, are available, or are soon to be available based on the estimated time of trip completion.
- **Proximity-Based Cab Suggestions:** When an employee searches for a cab, the system performs a quick geospatial query to identify cabs that are available and are in the vicinity of the employee's location. This would involve calculating the distance between the employee's location and the current locations of cabs from the real-time geospatial index.

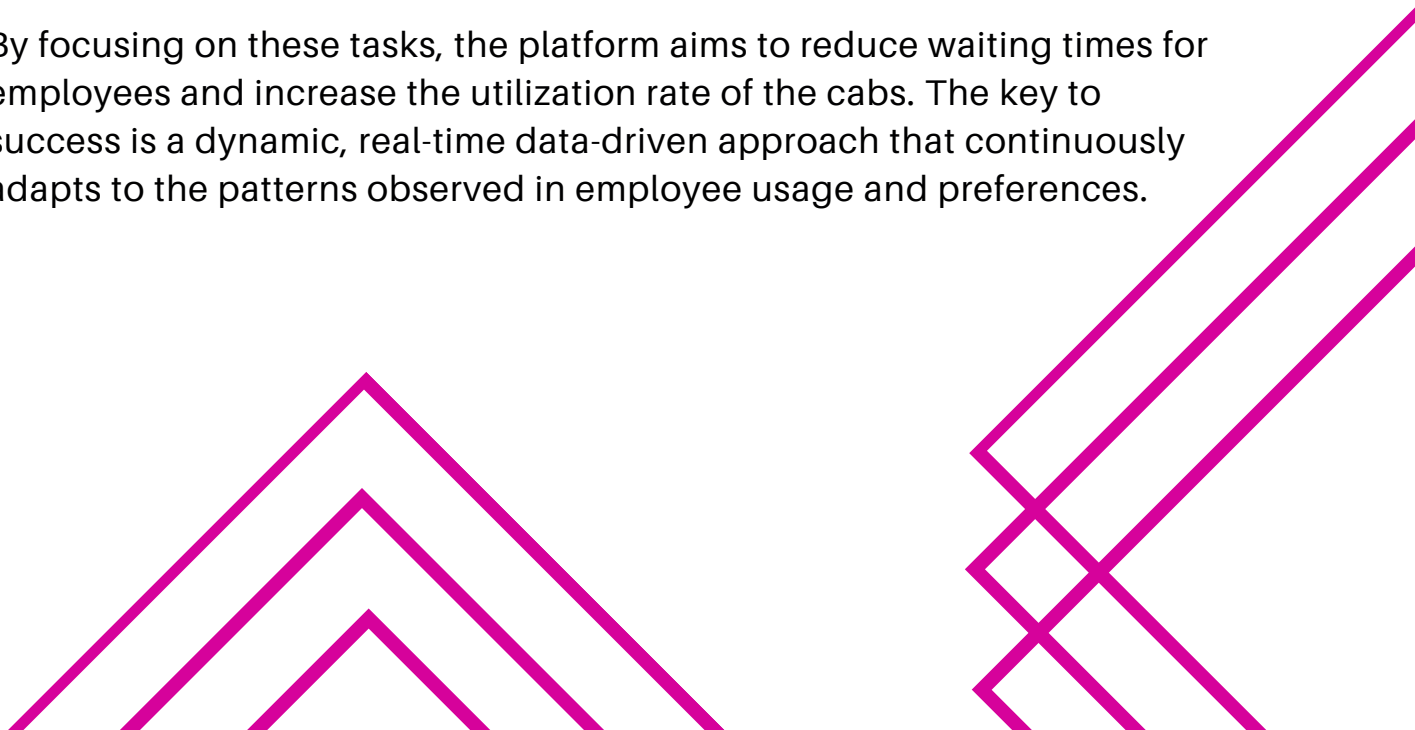
Algorithms:

- Upon receiving a ride request from a user, the client application communicates with the server to initiate the cab allocation process. The server processes the user's location to identify available cabs within a 500-meter radius using the spatial queries offered by Redis. We will get the coordinates of the nearest cabs from Redis and then use Google Maps API to sort them according to the time needed to reach the User's location. These cabs are then displayed to the user with the time needed to reach the user for selection, and simultaneously, the server communicates the ride request to the drivers of the selected cabs.
- If a driver accepts the request, the cab is allocated to the user, and the ride details are finalized. However, if there is no acceptance from any driver within one minute of the user's request, the algorithm enters a state of incremental search radius expansion. Specifically, the search radius is doubled, and the search for available cabs is retried.
- The server continues to double the search radius every minute until a driver accepts the request or until the maximum wait threshold of five minutes is reached. If this threshold is crossed without any driver acceptance, the algorithm concludes that no cabs are available within a reasonable distance. Consequently, the user is notified of the unavailability of cabs, ensuring timely communication.

Evaluating the System's Effectiveness

- **Response Time Measurement:** The system's effectiveness is measured by how quickly it can provide suggestions. The time from the employee's search request to the display of nearby cabs is logged and analyzed.
- **Relevance of Suggestions:** The relevance is gauged by how often the employees select the suggested cabs and the feedback provided after the ride. If employees frequently ignore suggestions or provide negative feedback, the relevance algorithm might need adjustments.
- **Feedback Loop Integration:** Employees can provide feedback on the cab suggestions, which the system can use to refine the suggestion algorithm. For example, if employees prefer cabs that will be available sooner, even if they are a bit farther away, the system can adjust its suggestions accordingly.
- **Algorithm Tuning:** The system regularly reviews the performance data and feedback to tune the proximity thresholds and the time frame for when cabs are expected to become available.
- **User Experience Metrics:** Metrics like the number of successful matches, average waiting time after selection, and user satisfaction scores provide insights into the system's performance.

By focusing on these tasks, the platform aims to reduce waiting times for employees and increase the utilization rate of the cabs. The key to success is a dynamic, real-time data-driven approach that continuously adapts to the patterns observed in employee usage and preferences.



Employing Caching

Incorporating caching into the Smart Cab Allocation System can significantly enhance its performance by reducing database load and improving response times. we can utilize caching effectively by:

- **Caching Real-Time Cab Locations:** Implement an in-memory data store like Redis to cache the real-time locations of cabs. This approach allows for quick access to current cab locations without repeatedly querying the database, which is crucial for real-time systems.
- **Caching User Session Data:** To improve user experience, especially in terms of authentication speed, user session tokens or credentials can be cached after the initial login. This reduces the need for frequent database hits for authentication checks on subsequent requests.
- **Caching Trip Histories and Analytics:** For quick access to historical data, caching trip histories and related analytics can be beneficial. This is particularly useful for generating reports or analytics without extensive database queries.
- **Cache Invalidation and Update Strategies:** Implement strategies for real-time cache updates when data changes and set appropriate Time-to-Live (TTL) for each cache entry. Ensure cache invalidation is handled properly to maintain data consistency.
- **Choosing the Right Caching Strategy:** Depending on the nature of the data, different caching strategies like write-through cache (for critical data) or lazy loading (for less critical data) can be employed.

Scalability and High Availability of Cache: The caching solution should be scalable and capable of handling high loads. A distributed caching system is recommended for high availability and to avoid a single point of failure.

By strategically implementing caching in these areas, the Smart Cab Allocation System can achieve faster data retrieval, reduced load on the database, and an overall more efficient and responsive service.

Thank You