Smart Cab Allocation System

Introduction

Smart Cab is a pioneering solution focused on optimising cab allocation for administrators and enhancing the user experience, and it employs cutting-edge algorithms and real-time data tracking. Smart Cab 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.

Project Overview

The Smart Cab Allocation System caters to three key user roles: Users, Drivers, and Administrators, each endowed with 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:

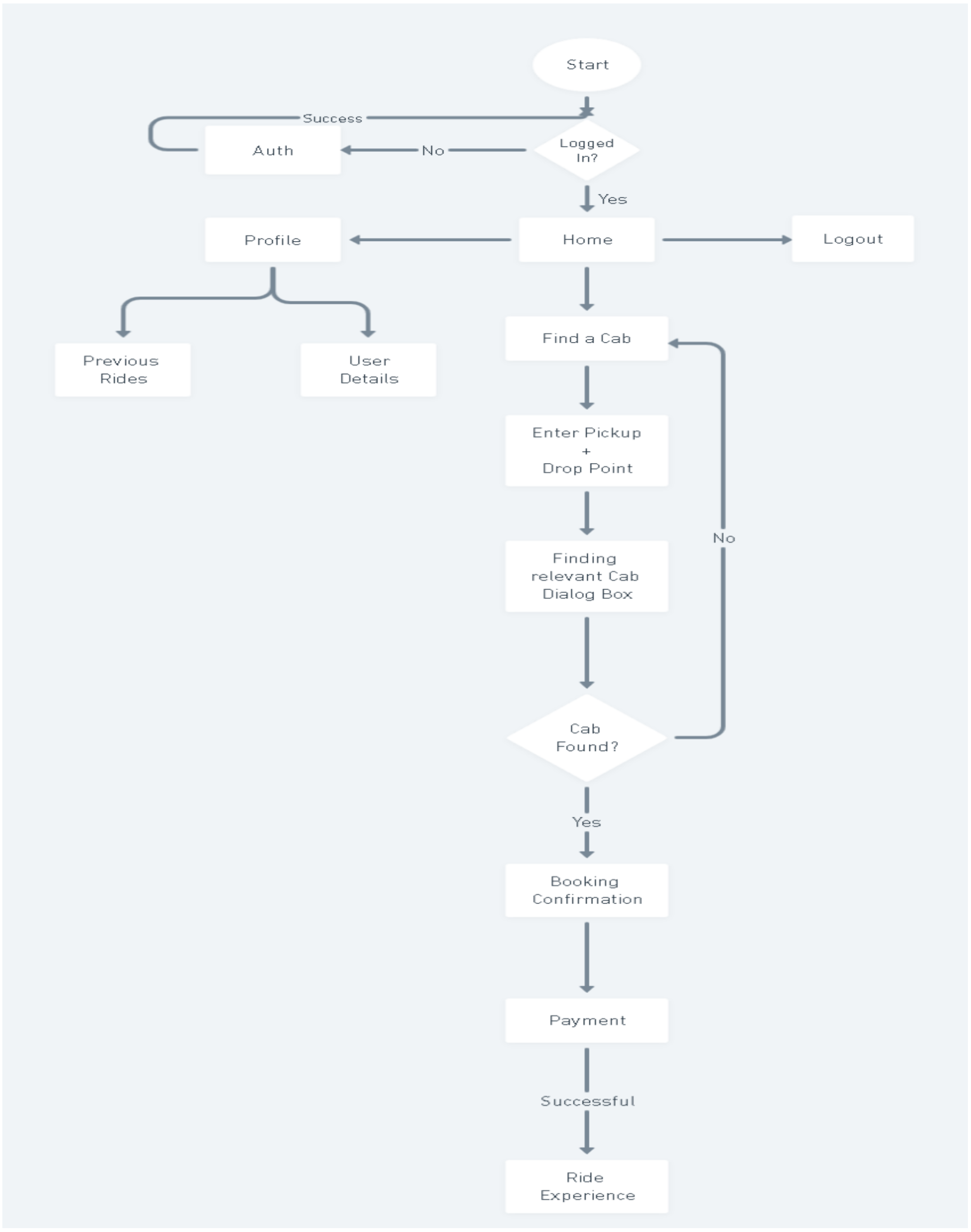
- Trip Acceptance and Rejection: Drivers have the autonomy to accept or reject trip requests based on their availability and location, ensuring flexibility in their schedules.
- Real-Time Navigation: The system provides drivers with real-time navigation assistance, optimising routes and estimated arrival 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: Administrators can optimise the cab allocation algorithm, ensuring efficient and strategic allocation of cabs to minimise overall travel distance.
- User and Driver Management: Administrators have the authority to manage user and driver profiles, including account creation or removal, ensuring a secure user base.
- Fault-Tolerant Mechanisms: Administrators are equipped with tools to implement fault-tolerant mechanisms, backup and recovery strategies, and error recovery procedures, contributing to system robustness and data integrity.

These functionalities collectively ensure a seamless and efficient experience for users, drivers, and administrators within the Smart Cab Allocation System.

Project Flow Chart



Authentication

The libraries and mechanisms mentioned in the flowchart are:

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.
- By utilising Bcrypt, the authentication process attains a heightened level of security, mitigating vulnerabilities associated with common password usage and potential brute-force attacks. The library's salting mechanism adds an extra layer of protection by preventing attackers from leveraging precomputed tables or rainbow tables.

authToken/JWT: JSON Web Token for Secure Identity Representation

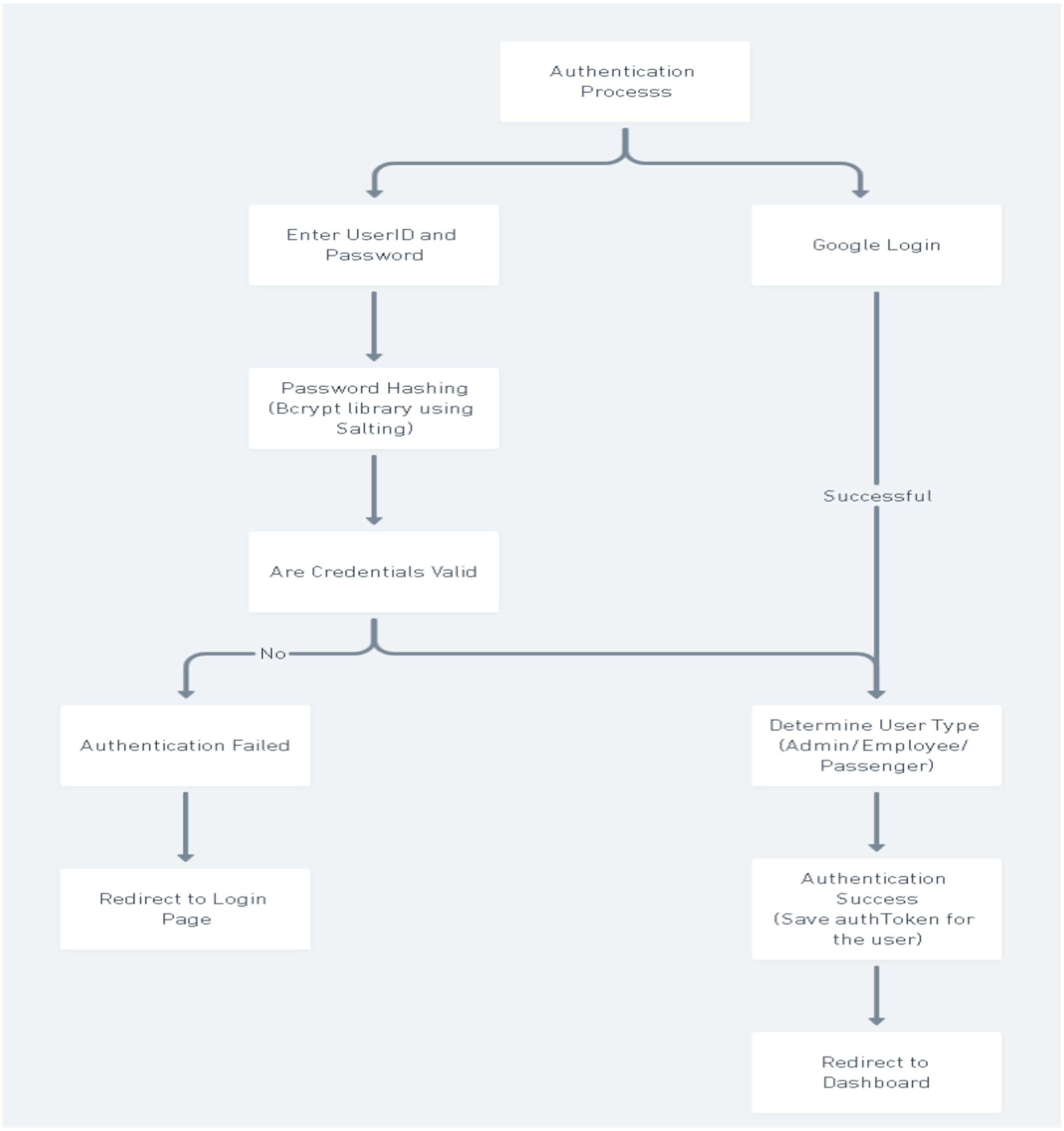
- The authToken, implemented as a JSON Web Token (JWT), serves as a secure and cryptographically signed representation of the user's identity. This token includes claims that convey essential user details and permissions, facilitating secure session management and subsequent user authentication for requests.
- 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 (Google Login): Secure User Authentication via Google Services

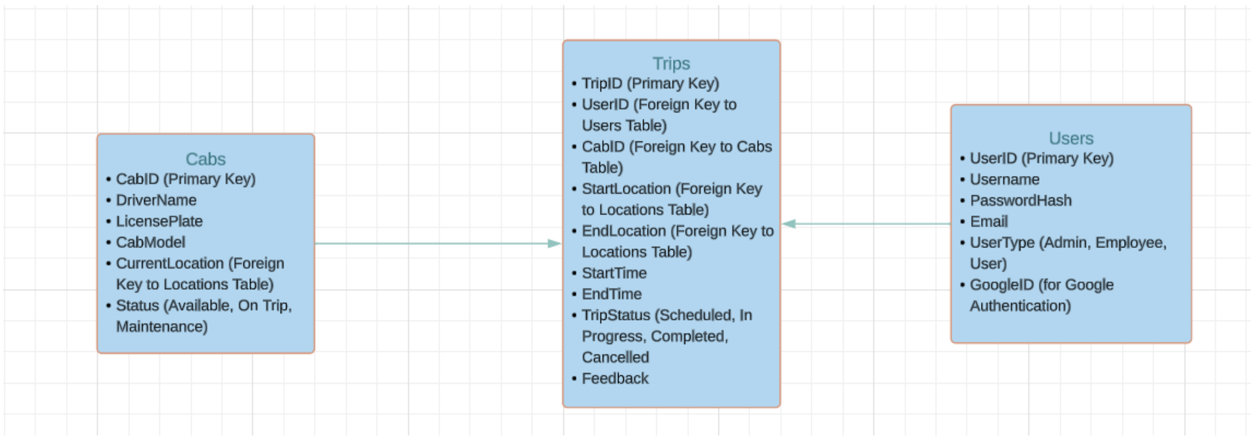
- OAuth 2.0 is employed as an authorisation framework that enables secure user authentication through Google's services. This mechanism grants the application limited access to user accounts on its HTTP service by delegating the authentication process to Google's hosting service.
- Leveraging OAuth 2.0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

In summary, the combination of Bcrypt, authToken/JWT, and OAuth 2.0 plays a pivotal role in creating a secure, efficient, and standardised authentication process. These libraries and mechanisms collectively contribute to the robustness of user identity management, password security, and the overall integrity of the authentication flow within the application.

Authentication Flow Chart


Authentication Processs
Enter UserID and Password
Google Login
Password Hashing (Bcrypt library using Salting)
Successful
Are Credentials Valid
No
Authentication Failed
Determine User Type (Admin/Employee/Passenger)
Redirect to Login Page
Authentication Success (Save authToken for the user)
Redirect to Dashboard

Database


Cabs
• CabID (Primary Key)
• DriverName
• LicensePlate
• CabModel
• CurrentLocation (Foreign Key to Locations Table)
• Status (Available, On Trip, Maintenance)
Trips
• TripID (Primary Key)
• UserID (Foreign Key to Users Table)
• CabID (Foreign Key to Cabs Table)
• StartLocation (Foreign Key to Locations Table)
• EndLocation (Foreign Key to Locations Table)
• StartTime
• EndTime
• TripStatus (Scheduled, In Progress, Completed, Cancelled
• Feedback
Users
• UserID (Primary Key)
• Username
• PasswordHash
• Email
• UserType (Admin, Employee, User)
• GoogleID (for Google Authentication)

Schema

Use REDIS for storing rows <CabID, location{x,y}> to support high read and write throughput for location data in real-time, and each cab would send its location every 10 seconds to update in the DB.

Redis would be useful here because of its Geospatial features and ability to partition data on the basis of location. To find the desired partition corresponding to a {lat, long}, use data structures like quadtrees.

Code for making tables in MongoDB:

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  userId: {type: String, unique: true},
  username: String,
  passwordHash: String,
  email: String,
  userType: String, // Admin, Employee, User
  googleId: String
});

const cabSchema = new mongoose.Schema({
  cabId: {type: String, unique: true},
  driverName: String,
  licensePlate: String,
  cabModel: String,
  currentLocation: { type: mongoose.Schema.Types.ObjectId, ref: 'Location'
},
  status: String // Available, On Trip, Maintenance
});

const tripSchema = new mongoose.Schema({
  tripId: {type: String, unique: true},
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  cabId: { type: mongoose.Schema.Types.ObjectId, ref: 'Cab' },
  startLocation: { type: mongoose.Schema.Types.ObjectId, ref: 'Location'
},
  endLocation: { type: mongoose.Schema.Types.ObjectId, ref: 'Location' },
  startTime: Date,
  endTime: Date,
  tripStatus: String // Scheduled, In Progress, Completed, Cancelled
});

const User = mongoose.model('User', userSchema);
const Cab = mongoose.model('Cab', cabSchema);
const Trip = mongoose.model('Trip', tripSchema);realTimeTrackingSchema);

// Connect to MongoDB
```

```
mongoose.connect('mongodb://localhost:27017/smartCabDB', {
useNewUrlParser: true, useUnifiedTopology: true });
```

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 a geospatial index, such as a **Quad-tree** or a database with geospatial capabilities like PostGIS, to update the cab's location. This index allows for efficient querying of location data.
- A dynamic list/map maintains a current view of each cab's status and other relevant details, ensuring that the system can quickly access and update the information as needed.

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 list/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, allowing for more cabs to be added without significant degradation in performance.
- Load balancers distribute the incoming data across multiple servers.
- The geospatial index and dynamic list/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. Here is an explanation of how this might work:

- **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 on a trip but 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.
- **Estimation of Cab Availability:** The system estimates when a cab will become available by using the current trip's destination and estimated time of arrival (ETA). If the ETA is within a certain time frame, say the next 5-10 minutes, the cab is flagged as a potential match for the employee.

Algorithm

1. 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 100-meter radius. These cabs are then displayed to the user for selection, and simultaneously, the server communicates the ride request to the drivers of the selected cabs.
2. 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.
3. 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 transparent and timely communication.

Pseudo Code

```
Begin Dynamic Cab Allocation

    Set initial search radius to 100 meters

    Set maximum wait time to 5 minutes

    Set time interval for radius expansion to 1 minute

    Start timer at user request


    While timer < maximum wait time

        Identify available cabs within the current search radius

        Display cabs to user and send request to drivers


        If a driver accepts within the time interval

            Allocate cab to user and finalize ride details

            Exit loop


        If no driver accepts and time interval is reached

            Expand search radius by doubling it

            Reset time interval


        If timer reaches maximum wait time without driver acceptance

            Notify user of cab unavailability

            Exit loop

End
```

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. Here's how caching can be effectively utilised:

- 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.