Cab Hailing Application

UCS2201 – Fundamentals and Practice of Software Development

A PROJECT REPORT

Submitted By

Srinidhi L	3122 225001 140
Sundaresh Karthic G	3122 225001 143
Surabhi Kamath	3122 225001 145

Department of Computer Science and Engineering

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

Kalavakkam – 603110

July 2023

Sri Sivasubramaniya Nadar College of Engineering

(An Autonomous Institution, Affiliated to Anna University)

BONAFIDE CERTIFICATE

Certified that this project report titled Cab Hailing Application is the bonafide work of Srinidhi Lakshmi Narayanan (3122 225001 140), Sundaresh Karthic Ganesan (3122 225001 143) and Surabhi Kamath (3122 225001 145) who carried out the project work in the UCS2201 – Fundamentals and Practice of Software Development during the academic year 2022-23.

Internal Examiner External Examiner

Date:

TABLE OF CONTENTS

Abstract

The report titled "Cab Hailing Application - Problem Statement" focuses on developing a software system to address the challenges in assigning cabs to customers based on their requests and locations. The objective is to provide an efficient and user-friendly cab hailing platform that ensures timely and convenient transportation for customers.

The system aims to calculate fares for customers based on various factors such as the base fare, distance travelled, peak demand, advance booking, and cancellations. These factors influence the overall fare structure and need to be implemented accurately.

To assign a driver to a customer, the system should consider multiple criteria. First, the driver assigned should minimize the waiting time for the customer. Second, the system should prioritize assigning the best-rated drivers available. Lastly, the assigned driver should travel the minimum distance to reach the customer's location.

The input for the system includes a set of customer requests, each comprising the location, destination, and preferred mode of travel (e.g., auto, mini sedan, sedan, SUV, Innova, etc.). Additionally, the system requires information on the number of available vehicles in each category.

The expected output of the system is the allocation of suitable cabs to customers, ensuring efficient matching based on their requests and available vehicles. Furthermore, the system should generate accurate bills for each customer, taking into account the relevant fare components.

By developing this cab hailing application, we aim to enhance the overall experience of customers and streamline the process of assigning cabs while considering factors such as waiting time, driver ratings, and distance optimization.

1. Problem Statement

Develop a software system for assigning cabs for customers based on their requests and locations. The customers are charged a fixed base fare plus fare based on the distance travelled. During peak demand time, a surge fee also will be charged. Apart from this, if the customer books the vehicle in advance, advance booking fees will be charged. If the customer cancels the ride for some reason, a cancellation fee will be charged.

Constraints

Assign a driver so that

- ➢ the customer can be picked up with minimum waiting time
- ➢ the best among the available drivers based on the average rating is assigned

The assigned driver should be driving the minimum distance to pick up the customer.

Input

Set of customer requests, where customer request comprises location, destination and mode of travel such as auto, mini sedan, sedan, SUV, Innova and so on.

Number of vehicles available in each category

Output

Allocation of suitable cabs to customers and bill generation

2. Extended Exploration

1. Introduction:

The cab hailing industry has witnessed significant growth in recent years, with customers increasingly relying on mobile applications to book rides conveniently. This extended exploration focuses on developing a comprehensive software system for a cab hailing application to address the challenges faced by customers and provide efficient and user-friendly transportation services.

2. Deep Diving into the Problem Statement:

The primary objective is to develop a cab hailing application that assigns suitable cabs to customers based on their requests and locations. The system should consider various factors, including customer preferences, available vehicles, distance optimization, fare calculation, and efficient driver allocation.

3. Key Features and Functionality:

The cab hailing application should incorporate the following key features and functionalities:

3.1 Customer Registration and Ride Booking:

- Customers should be able to register their profiles, providing necessary details such as name, contact information, and payment preferences.

- The application should allow customers to book rides by specifying their current location, desired destination, and preferred mode of travel (auto, mini sedan, sedan, SUV, etc.).

- Advanced booking functionality should be available for customers who wish to schedule rides in advance.

3.2 Vehicle Availability and Assignment:

- The system should maintain real-time information about the number and types of vehicles available in each category.

- Based on customer requests, the application should assign the most suitable available vehicle, considering factors such as distance, customer preference, and vehicle type.

- The assigned vehicle should minimize the waiting time for the customer and optimize the driver's distance to reach the customer's location.

3.3 Fare Calculation:

- The fare calculation algorithm should consider multiple components, including the base fare, distance travelled, peak demand surcharges, advance booking fees, and cancellation charges.

- The system should accurately calculate fares based on these factors and provide customers with transparent and itemized bills.

3.4 Driver Allocation:

- The application should maintain a database of registered drivers, including their profiles, availability, and average ratings.

- When assigning a driver to a customer, the system should prioritize the best-rated drivers who are available and can reach the customer with minimal travel distance.

- Customer feedback and driver ratings should be incorporated to ensure a quality experience for customers.

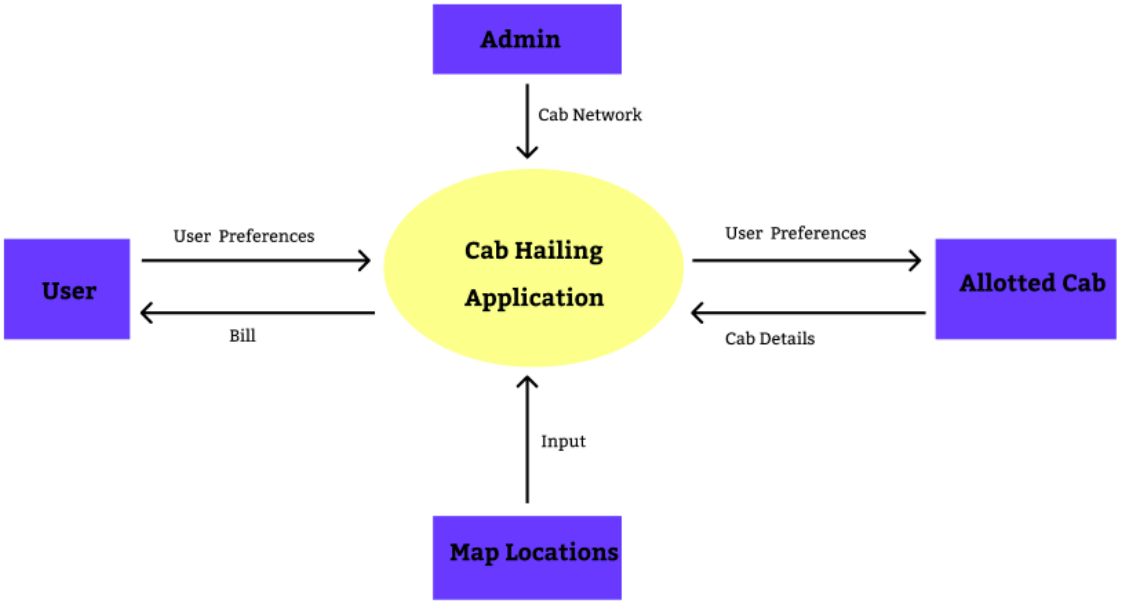
5. Conclusion:

Developing a robust cab hailing application that addresses the challenges associated with assigning cabs to customers is crucial in today's transportation landscape. By considering factors such as efficient vehicle allocation, fare calculation, driver assignment, and user experience, the application aims to provide customers with a seamless and convenient transportation service. Through this extended exploration, we have outlined the key features and functionalities that the cab hailing application should incorporate to deliver a comprehensive and satisfying user experience.

3. Project Analysis

To set a background, the context diagram has been attached. Following this are the Data Flow Diagrams (DFDs) built in the initial stages of developing this application.

3.1 Level 0 - Context diagram:



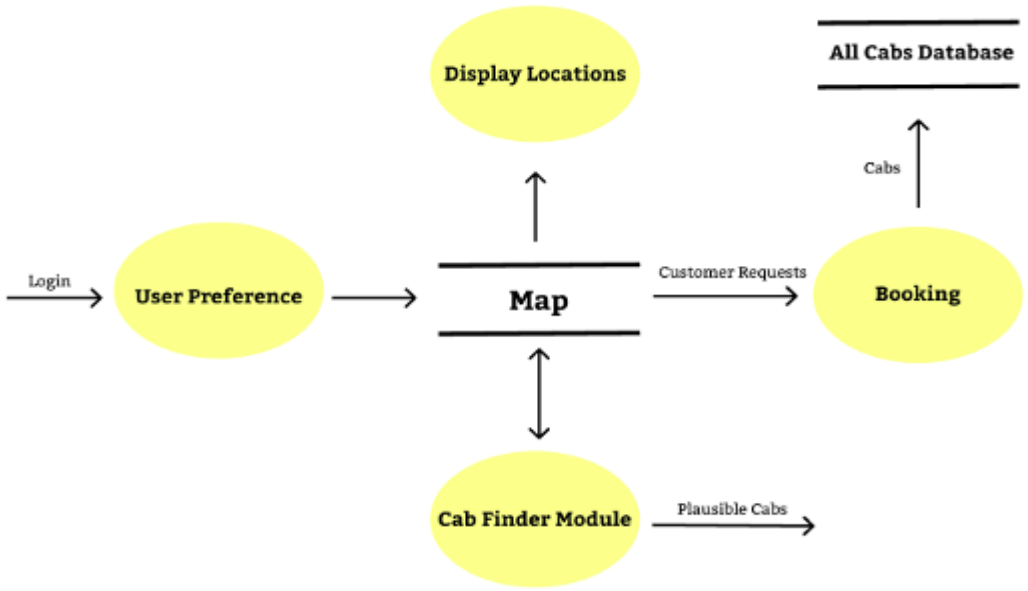
User: They are the primary beneficiaries of the service, and they input their preferences.

Admin: Administrators have the authority to monitor user activities, ensure cab availability, adjust pricing, and undertake various administrative tasks to maintain a seamless operation. They create a cab system.

Allotted Cab: The cab allocation process involves selecting the most suitable cab based on the user's preferences and requirements. The relevant cab details are then transmitted back to the application for the user's convenience.

Map Integration: The application harnesses geographical data to enhance the overall user experience. The map functionality provides information for both users and drivers, enabling smooth navigation, and efficient route planning.

3.2 Level 1- DFD



User Preferences: Upon logging in, users can specify their preferred vehicle type, allowing them to tailor their cab-hailing experience to their liking.

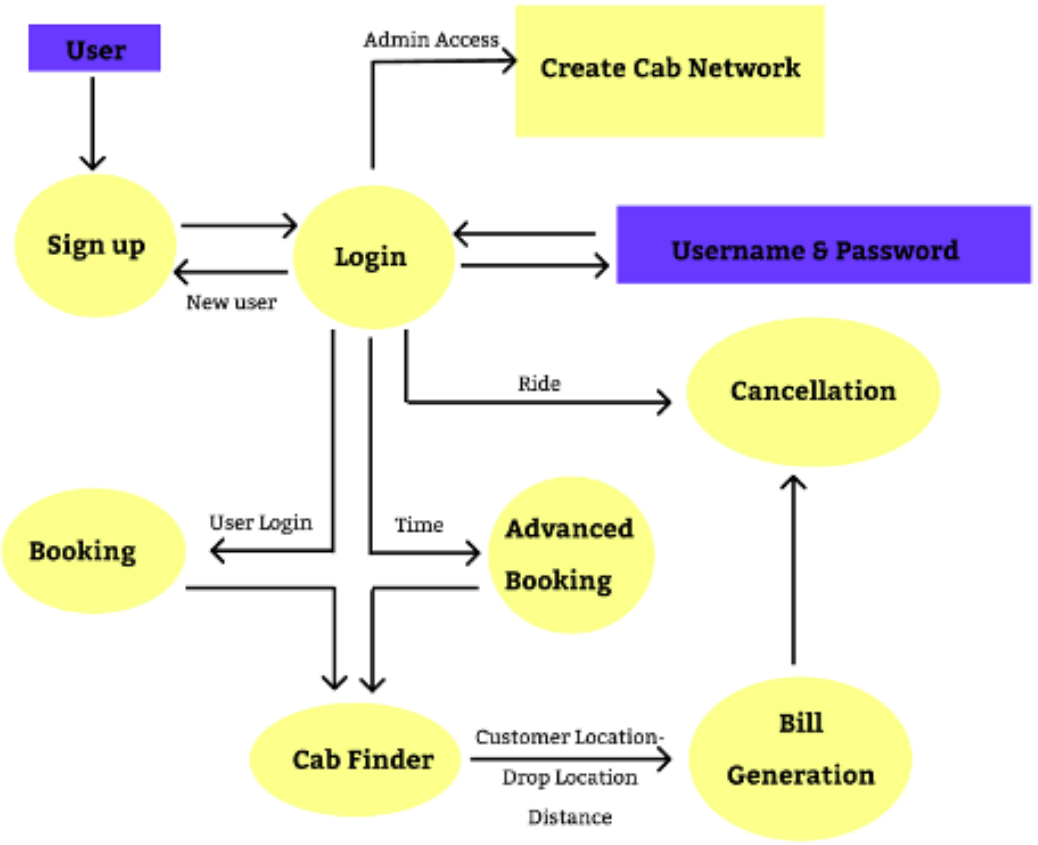
Map Integration: Leveraging an interactive map interface, users can effortlessly select their current location and desired drop-off point, ensuring accurate and efficient cab allocation.

Location Display: The application showcases a comprehensive display of various locations within the map, enabling users to explore and choose their destinations conveniently.

Cab Finder Module: Empowered by intelligent algorithms, the cab finder module utilizes user-provided data, including preferences, ratings, and proximity, to identify and recommend the optimal cab options available. This ensures a personalized and efficient cab allocation process.

Booking Process: Once the most suitable cab is determined based on the processed data, it is seamlessly reserved for the user. The cab's details are then added to the centralized allCabs database, containing comprehensive information about all available cabs within the system.

3.3 Level 2- DFD



1. Sign-up: This process involves new users registering themselves with the cab hailing system by providing necessary information. Once the sign-up process is completed, the user's data is stored in the system for future reference.

2. Login: The login process allows users who have already signed up to access their accounts by providing their unique credentials (e.g., username and password). Upon successful authentication, the system grants the user access to their personalized account features and functionality.

3. Booking: This process involves a user selecting their desired cab type, providing their current location and destination details, and requesting a ride through the cab hailing application. The system receives the booking request and proceeds to allocate an appropriate cab for the user.

4. Advanced Booking: This process enables users to specify the date, time, and other relevant details for future rides. The system stores the advanced booking information and ensures that the requested ride is allocated at the scheduled time.

5. Cab Finder Wizard: It takes into account factors such as cab type, availability, proximity, and user preferences to recommend the best cab options. The system utilizes the cab finder wizard process to present the user with a curated selection of available cabs.

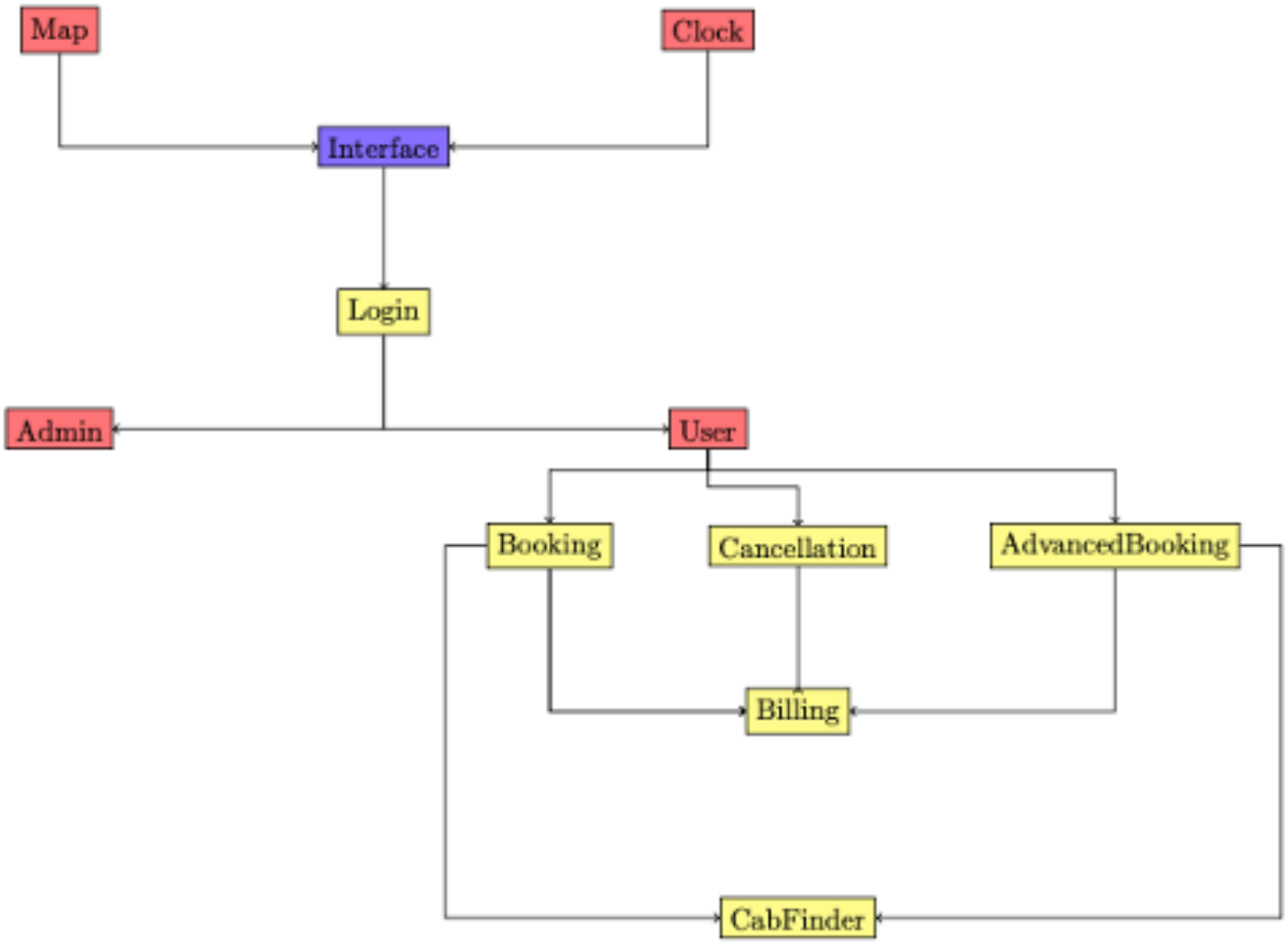
6. Bill Generation: After a ride is completed, the system calculates the fare based on factors like distance travelled, duration, and any additional charges.

7. Cancellation: In cases where a user needs to cancel a previously booked ride, the cancellation process allows them to do so.

4. Detailed Design

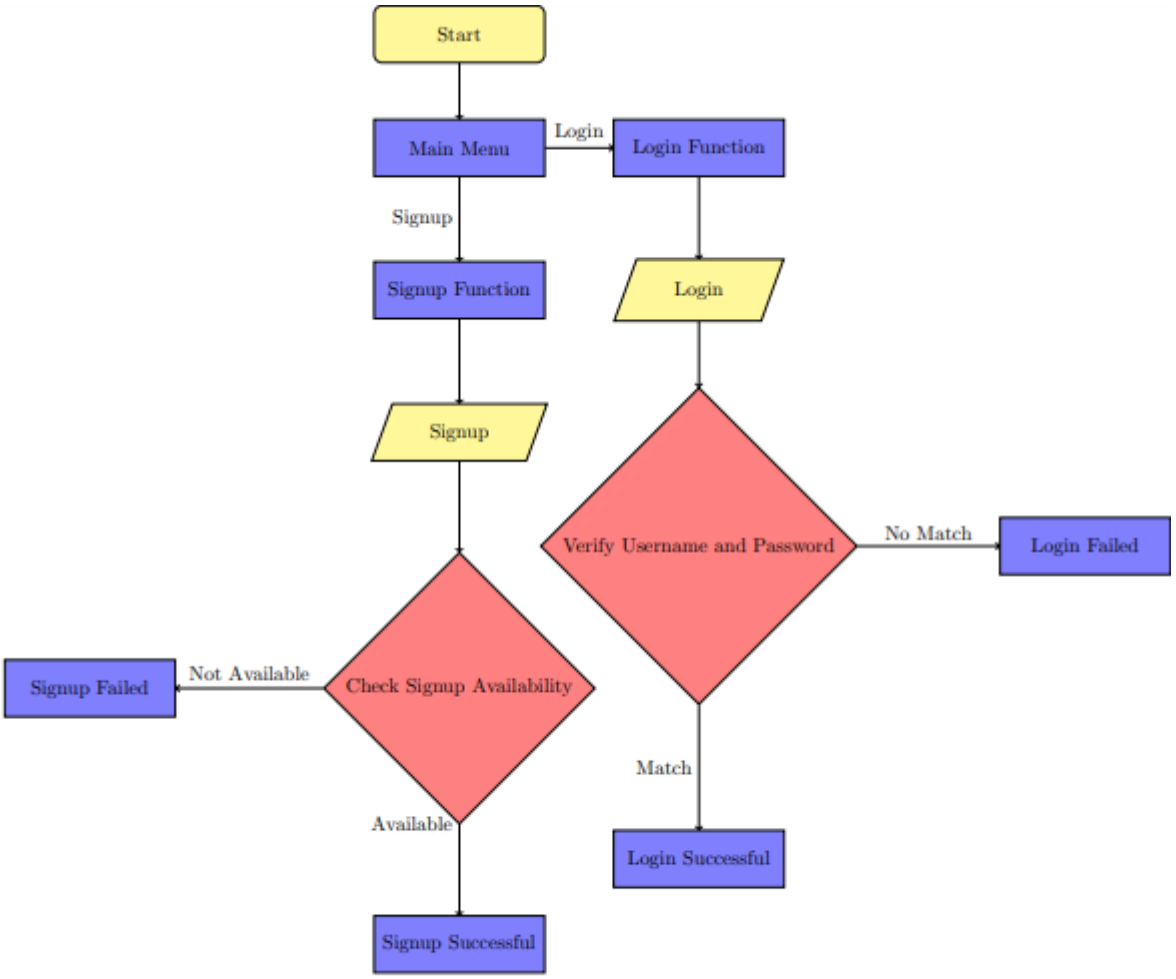
The detailed design, including the overall architecture diagram, Structure Chart/ Flow Chart for individual Modules are as follows.

4.1 Architecture Diagram

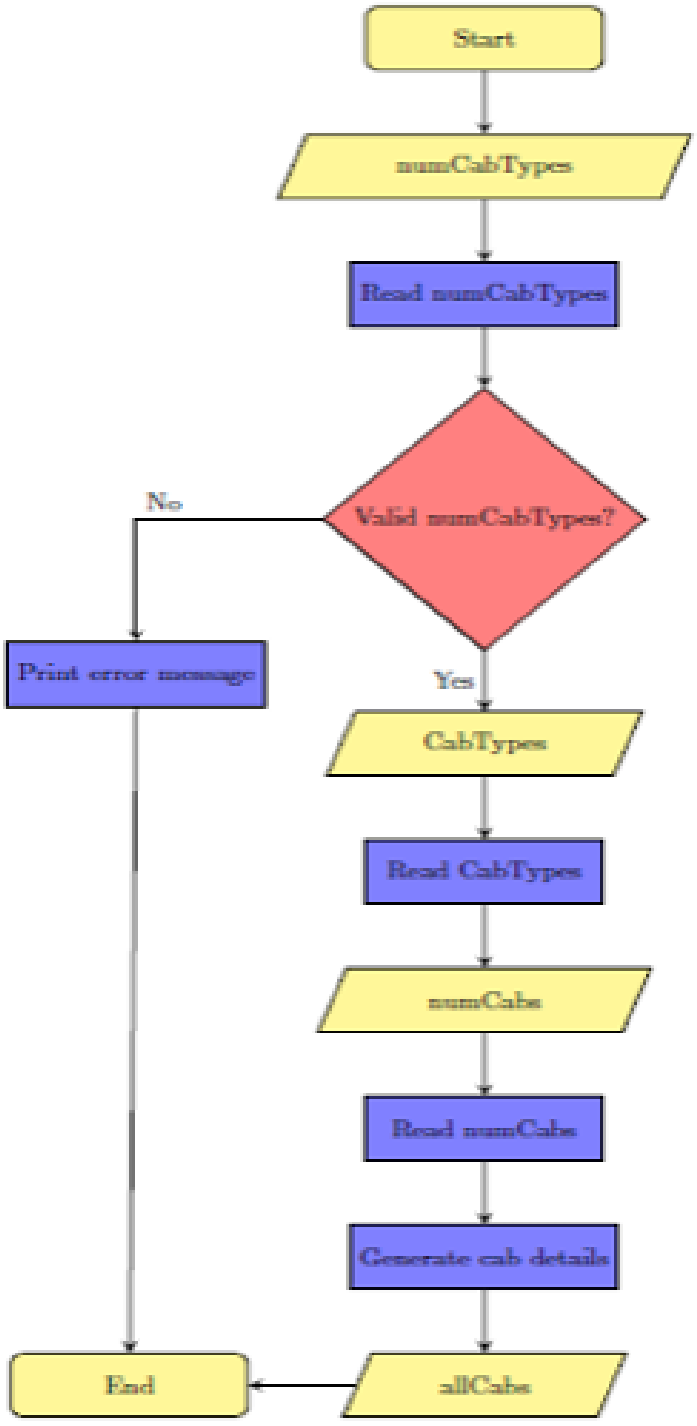


4.2 Flowcharts for individual modules:

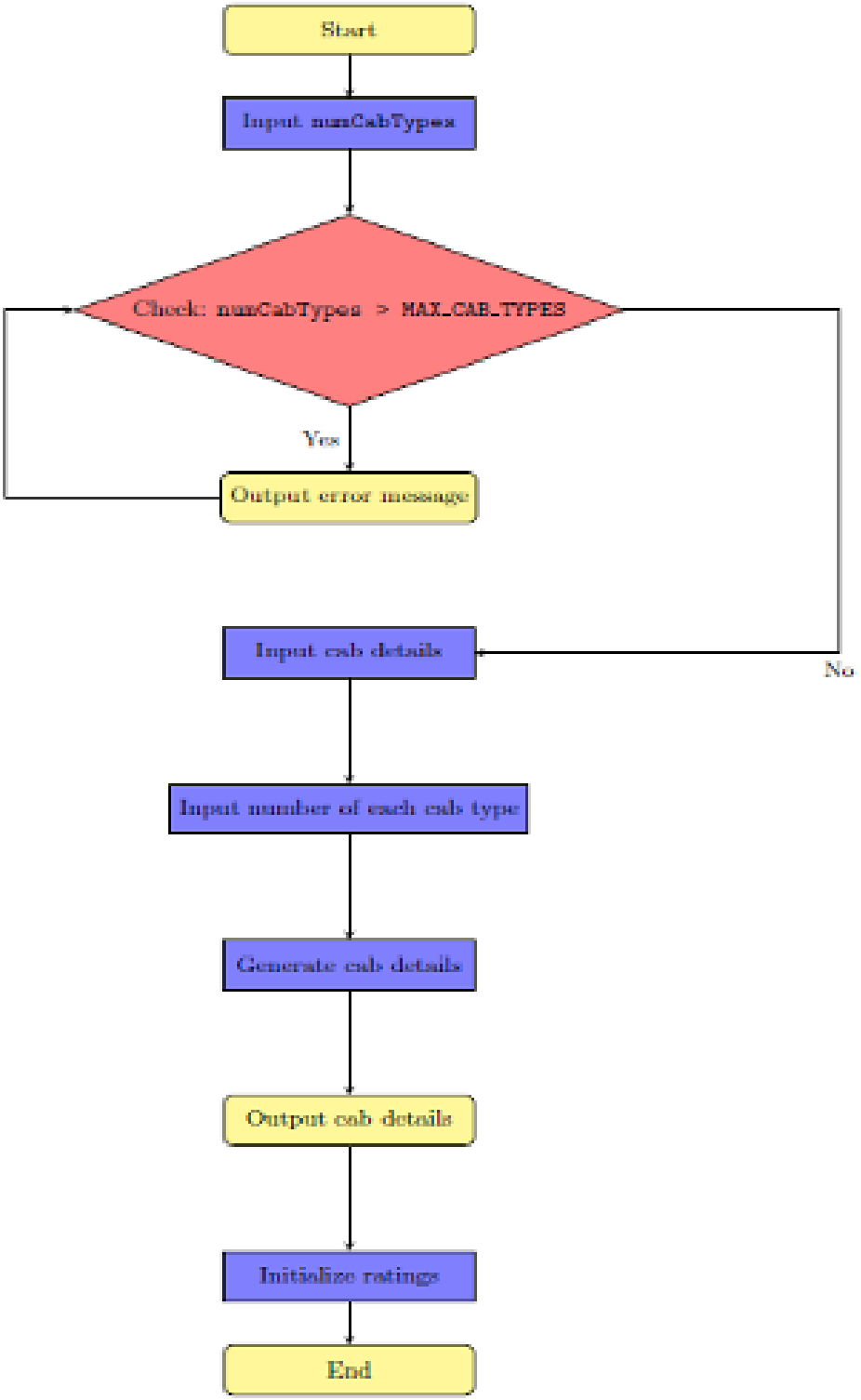
4.2.1 Login Module



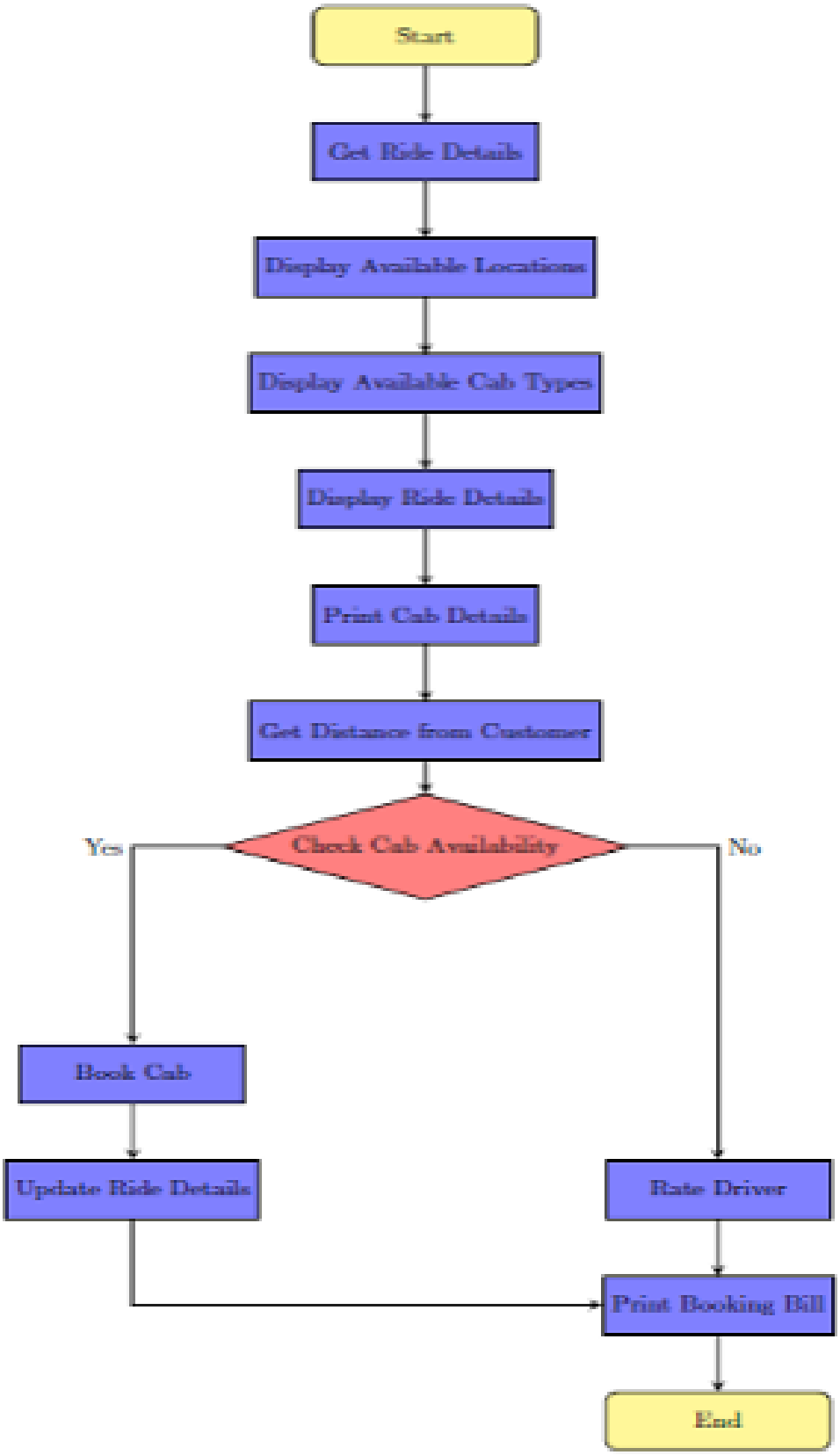
4.2.2 Admin Module



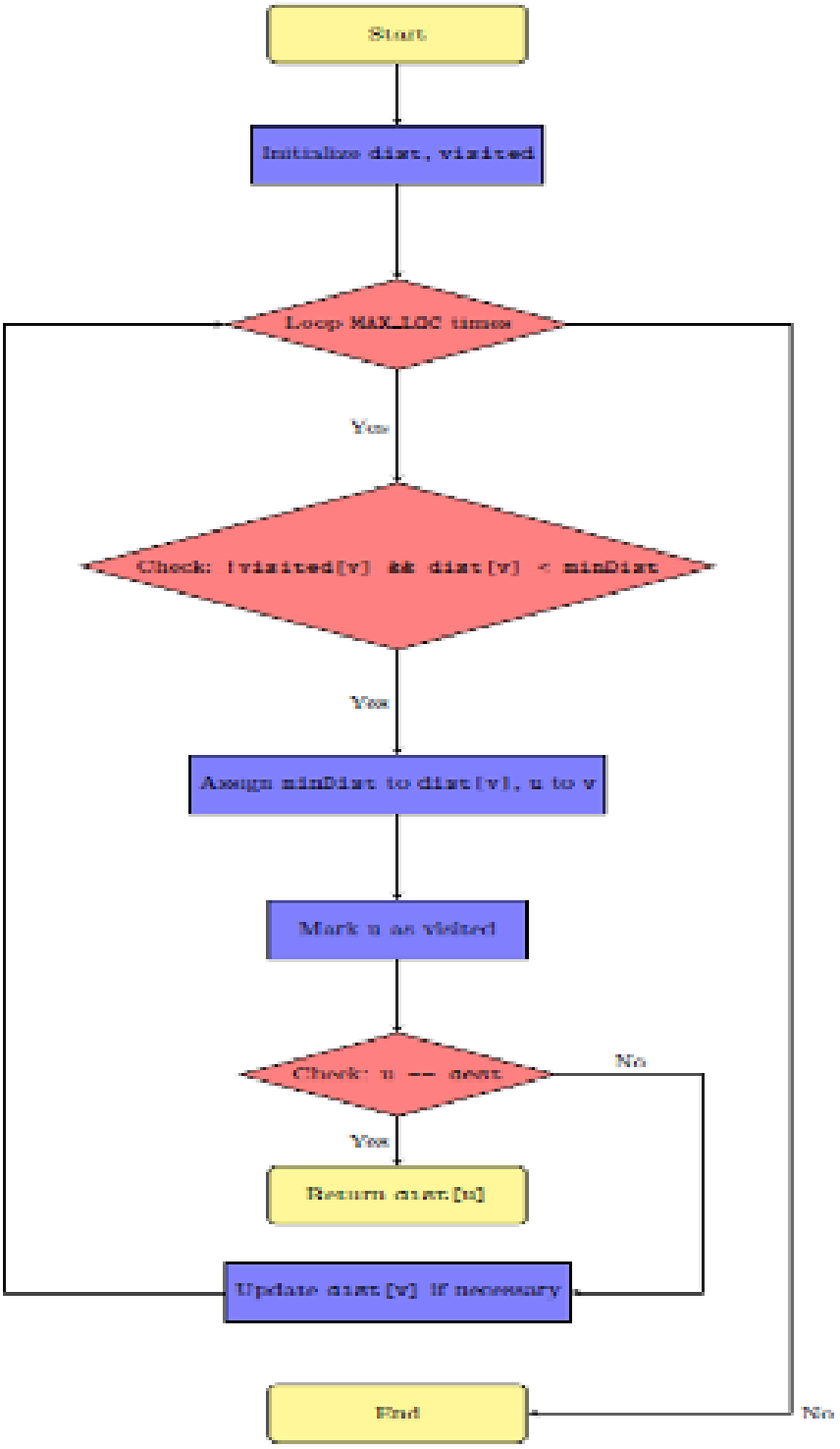
4.2.3 Generate cab System



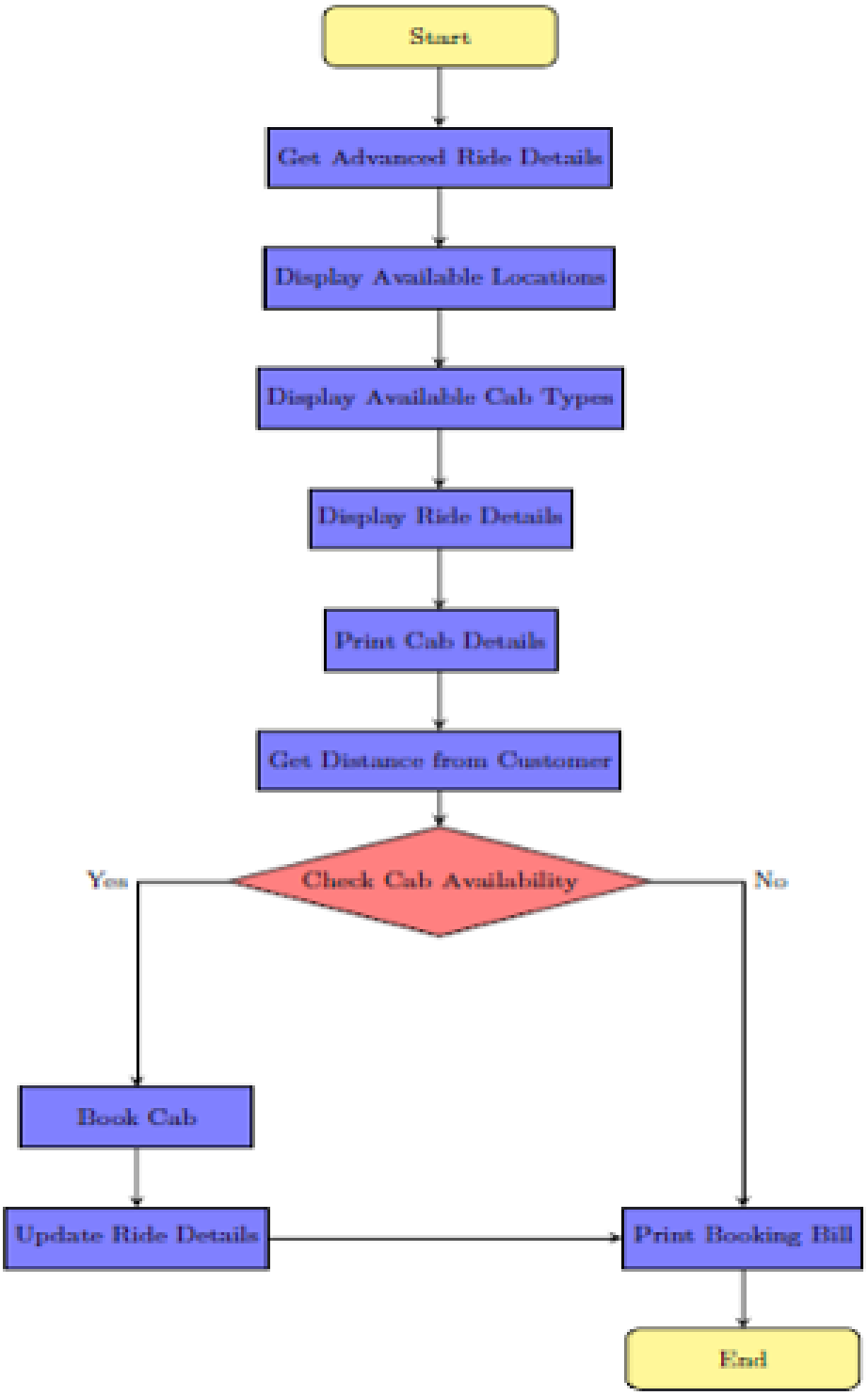
4.2.4 Booking Module



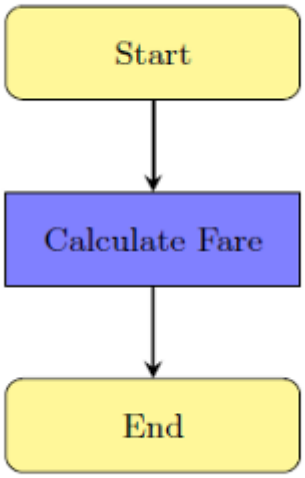
4.2.5 Customer to drop location (calculates shortest distance between pickup and drop location)



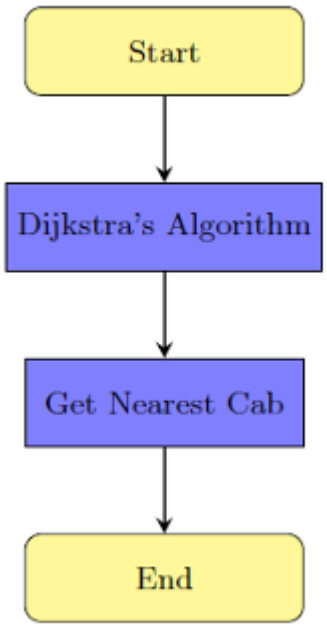
4.2.5 Advanced Booking



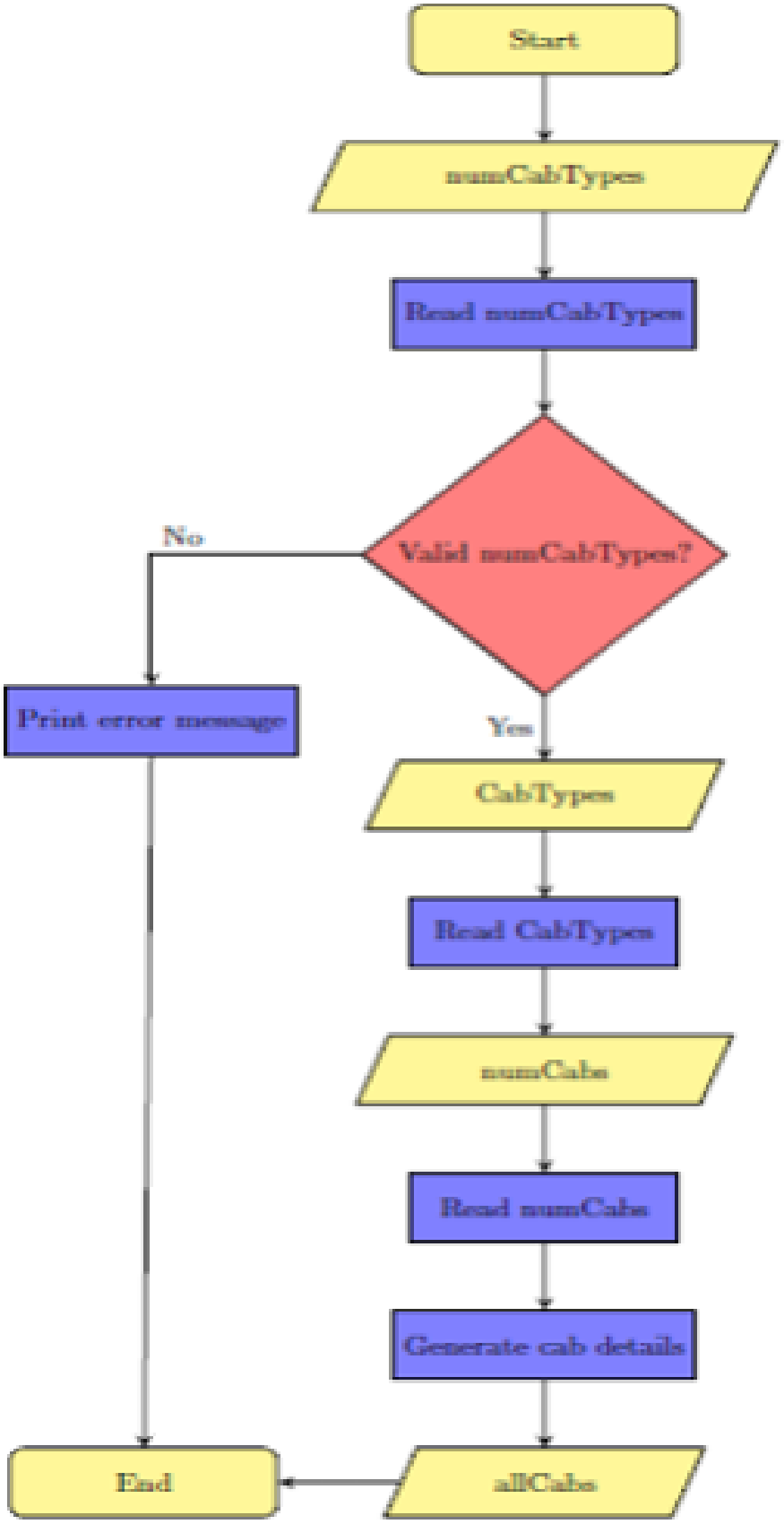
4.2.6 Billing Module



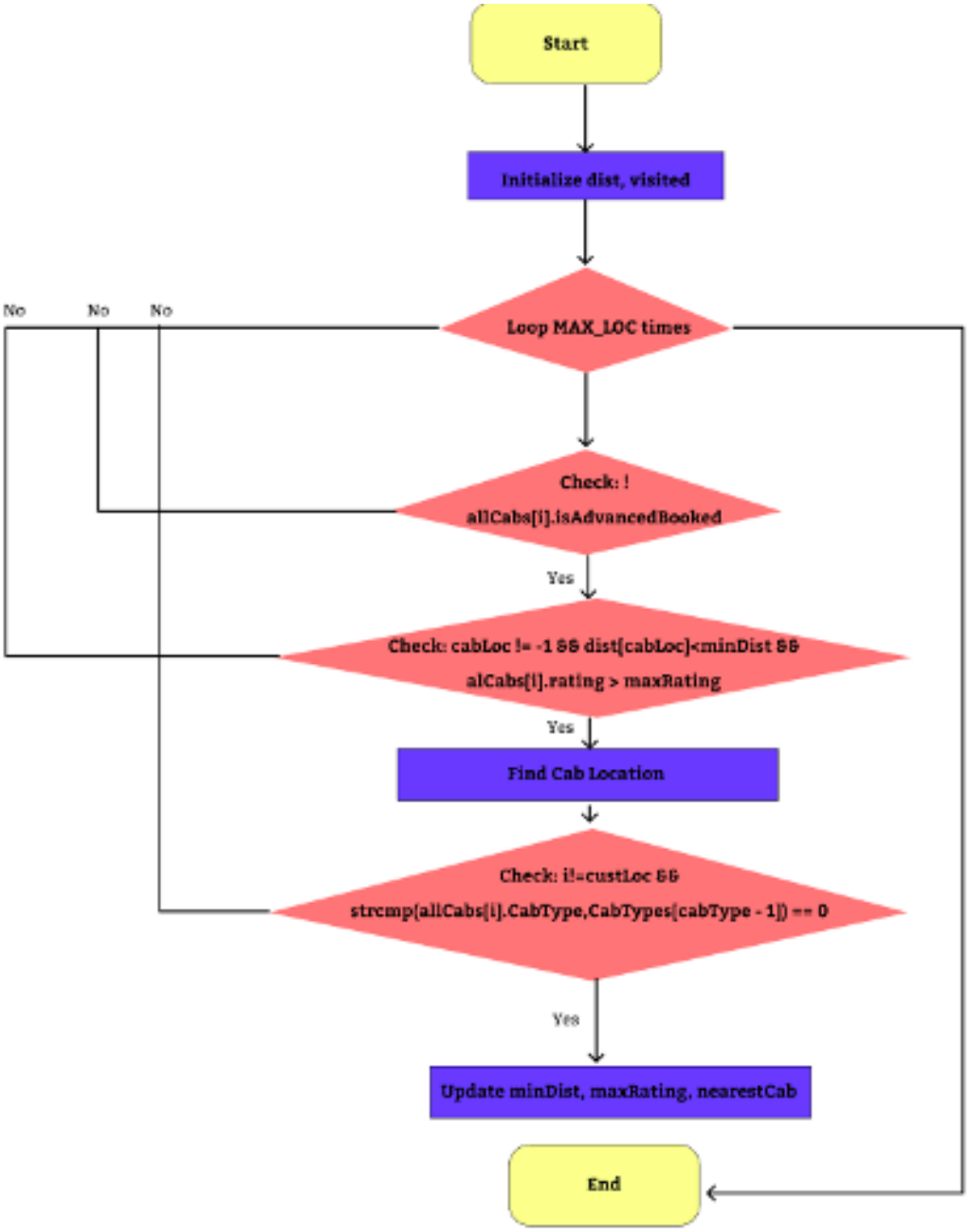
4.2.7 Cab Finder Module



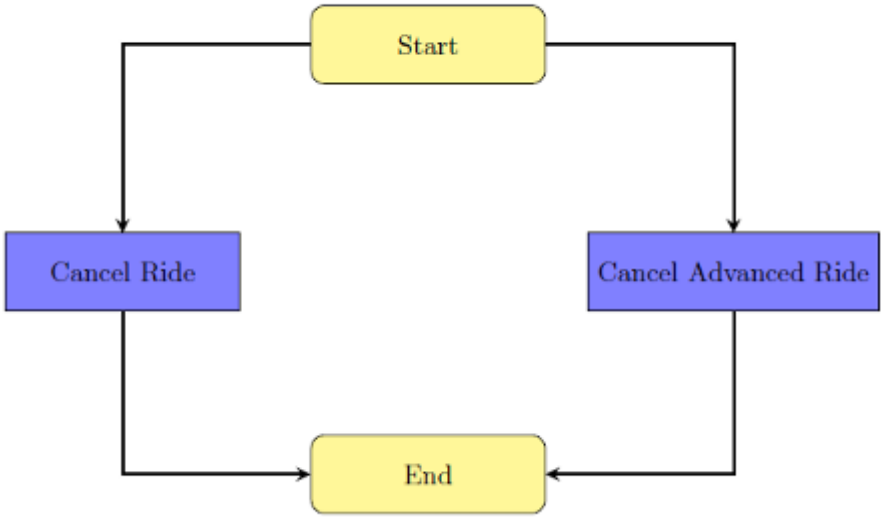
4.2.8 Dijkstra's Algorithm



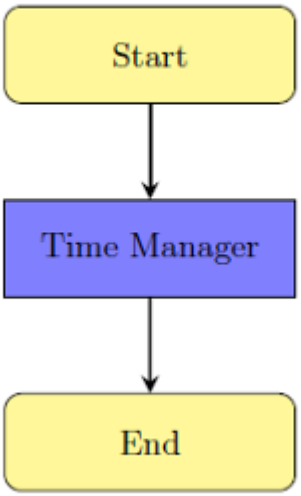
4.2.9 Get Nearest cab



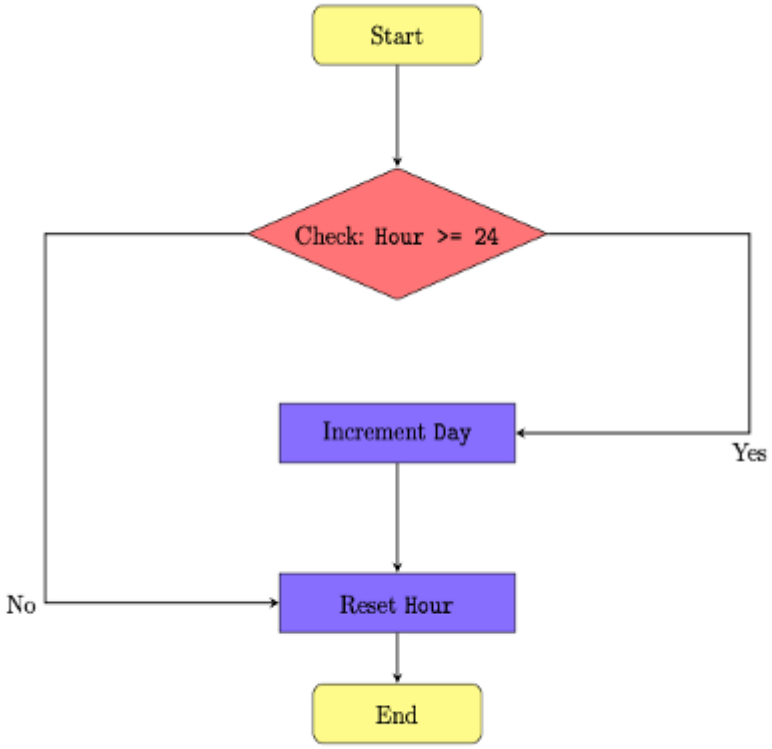
4.2.10 Cancellation



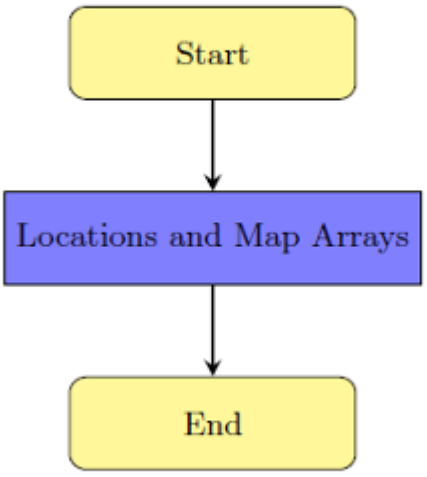
4.2.11 Clock



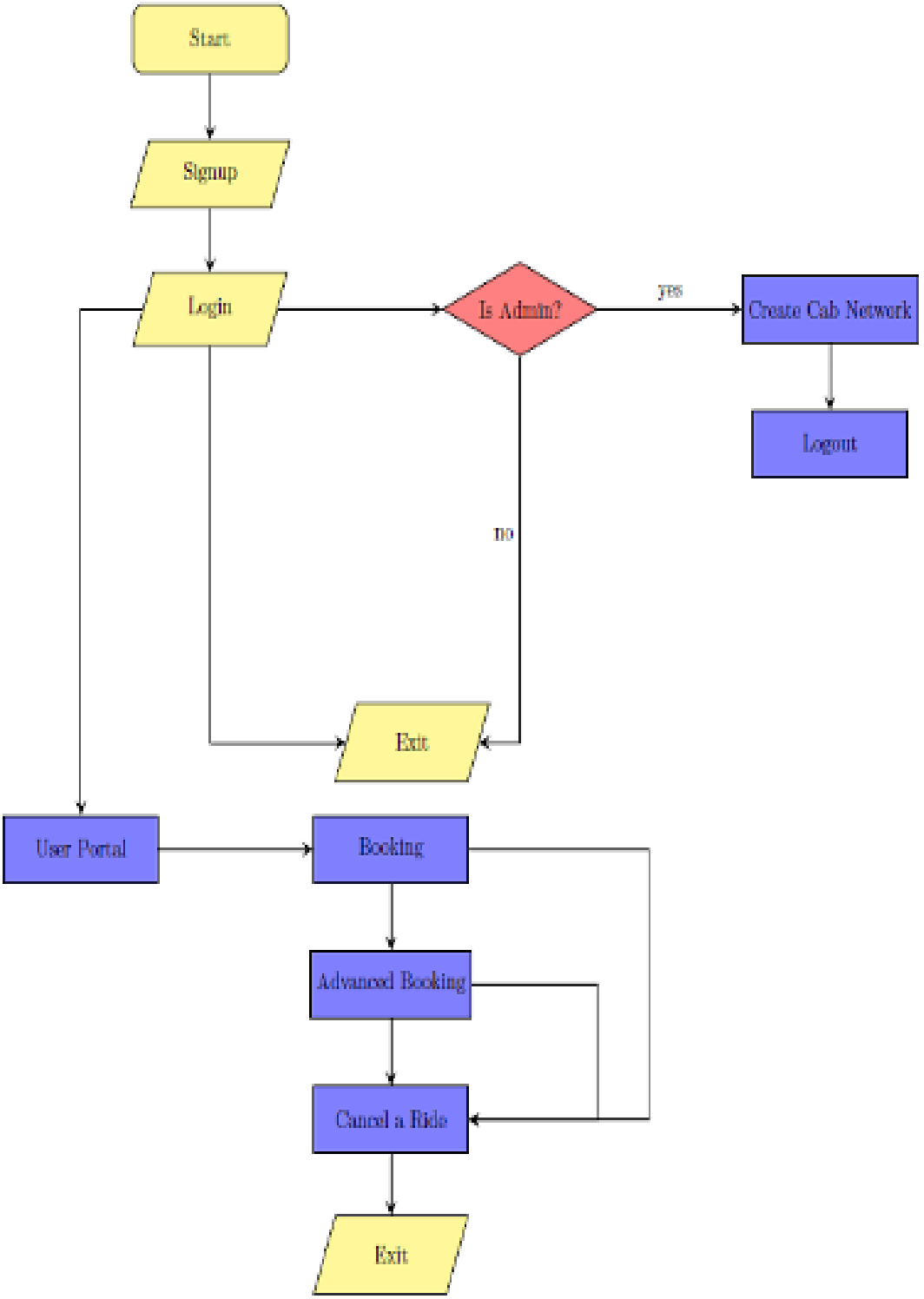
4.2.12 Time Manager Flowchart



4.2.13 Map Database



4.2.14 Interface module:



5. Module Description

5.1 Login module:

5.5.1.User Structure: The module defines a structure User that represents a user with a username and password. It has a maximum length for both username and password fields.

5.1.2.User Data Storage: The module declares an array of users to store user data. It can hold up to 20 user records. Each user record is of type User structure.

5.1.3.Signup Function: The module provides a signup() function that allows a user to sign up by entering a username and password. It checks if the maximum number of users has been reached and prevents signup if so. It also checks if the entered username is already taken and prompts for a different username. The user's information is stored in the users array.

5.1.4.Login Function: The module provides a Login() function that allows a user to login by entering their username and password. It checks if the entered username and password match with any user record in the users array. If a match is found, the user is considered logged in, and their username is stored in the current_user variable.

5.1.5.Successful Signup and Login: When a user successfully signs up or logs in, a corresponding success message is displayed to the user.

5.1.6.Return Values: The Login() function returns an integer value to indicate the login status. It returns 1 if the login is successful and 0 if the login fails.

5.2 Admin Module:

1. It defines the data structure `Cab` to represent a cab, including its type, license ID, default location, current location, rating, booking status, and requested day.

2. It declares the array `allCabs` to store information about multiple cabs.

3. It provides a function `generateLicenseId` to generate a random license ID for a cab.

4. It allows the administrator to input details about different cab types, such as their names and quantities, and generates the cab details accordingly.

5. The module also prints the generated cab details, including cab type, license ID, default location, and current location.

6. It initializes the rating for all cabs and provides functionality to manage the booking status of normal and advanced bookings.

Overall, the `admin.h` module handles the administration and generation of cab details for the cab booking system.

5.3 Generate Cab Systems:

1. The function starts by prompting the user to enter the number of cab types (`numCabTypes`) available in the system.

2. If the entered number of cab types exceeds the maximum allowed (`MAX_CAB_TYPES`), an error message is displayed, and the function returns `1` to indicate an error.

3. Next, the function asks for the details of each cab type. It prompts the user to enter the name of each cab type and stores them in the `CabTypes` array.

4. After collecting the cab type names, the function proceeds to ask the user for the number of cabs available for each cab type. The number of cabs for each type is stored in the `numCabs` array.

5. Using the collected information, the function generates the details for each cab. It iterates over each cab type and for each type, generates the specified number of cabs.

6. For each generated cab, a license ID is generated using the `generateLicenseId` function. The cab's default location and current location are assigned randomly from the `Locations` array.

7. Other details such as cab type, rating, advanced booking status, and requested day are initialized for each cab.

8. Once all the cab details are generated, the function prints the details of each cab, including the cab type, license ID, default location, and current location.

9. Finally, the function sets an initial rating of 4.5 for all cabs.

10. The function returns `0` to indicate successful execution.

The `generateCabSystem` function is responsible for initializing the cab system by collecting user input for cab types and the number of cabs for each type. It then generates and assigns details to each cab, including license ID, locations, and other attributes.

5.4 Booking Module

5.4.1.Ride Structures: The module defines two structures, Customer_Requestand Ride_Details, to represent ride-related information. Customer_Request stores the customer's location, drop location, and preferred cab type. Ride_Details stores the same information along with the allotted cab index, driver rating, and ride ID.

5.4.2.Display Functions: The module provides functions to display available locations and cab types. displayLocations() and displayCabTypes() print the available options for the user to choose from.

5.4.3.Ride Details Functions: The module provides functions to display and print ride details. displayRideDetails() prints the details of a customer's ride request. printCabDetails() prints details about the allotted cab, including cab type, license ID, current location, and distance from the customer's location.

5.4.4.Booking Bill Functions: The module includes functions to print the booking bill. printBookingBill() calculates and prints the fare, surge fare (if applicable), and total fare for a ride. It also displays the customer's location, drop location, driver rating, ride ID, and a thank you message.

5.4.5.Input Functions: The module includes a function getRideDetails() to retrieve ride details from the user. It prompts the user to enter the customer location, drop location, and preferred cab type. The function returns a Customer_Request structure with the entered details.

5.4.6.Ride Management Functions: The module includes functions for ride management. CustomerToDrop() calculates the shortest distance between a customer's location and drop location using a graph representation. generateRideID() generates a

random ride ID. updateRideDetails() updates the ride details after cab allocation, including the cab index, driver rating, and ride ID. rateDriver() prompts the user to rate the driver and returns the rating.

5.5 Customer to drop location:

The `CustomerToDrop` function calculates the shortest distance between a source location and a destination location in a graph using Dijkstra's algorithm.

1. Initialize variables: Initialize the `dist` array to store the shortest distances from the source location, and the `visited` array to track visited locations. Set all distances to `INT_MAX` except for the source location, which is set to 0.

2. Iterate through locations: Repeat the process for a maximum of `MAX_LOCATIONS - 1` times.

3. Find the closest unvisited location: Find the unvisited location (`u`) with the minimum distance (`minDist`) from the source.

4. Mark the location as visited: Set `visited[u]` to 1 to mark the location as visited.

5. Check if the destination is reached: If the current location `u` is the destination, return the shortest distance stored in `dist[u]`.

6. Update distances: Update the distances of neighboring unvisited locations (`v`) from `u`. If the distance from the source to `u` plus the weight of the edge between `u` and `v` is smaller than the current distance to `v`, update `dist[v]` with the new shorter distance.

7. No path found: If the destination is not reached after iterating through all locations, return -1 to indicate that there is no path between the source and destination.

5.6 Advanced Booking Module

1. It defines the data structure `Advanced_Customer_Request` to represent the advanced booking details, including customer location, drop location, cab type, requested day, and requested hour.

2. It defines the data structure `Advanced_Ride_Details` to represent the details of an advanced booked ride, including customer location, drop location, cab type, allotted cab index, rating, requested day, requested hour, and ride ID.

3. It provides functions to display available locations and cab types for advanced booking.

4. It allows the user to input advanced ride details, including customer location, drop location, preferred cab type, and requested day and hour.

5. It calculates and returns the shortest distance from the customer location to the drop location using Dijkstra's algorithm.

6. It updates the ride details after cab allotment, including the allotted cab index, rating, current location, requested day, requested hour, and ride ID.

7. It prints the details of the allotted cab, including cab type, license ID, current location, and distance from the customer.

8. It prints the advanced booking bill, including customer location, drop location, driver rating, ride ID, distance, requested day, requested hour, fare, surge fare (if applicable), and total fare.

9. It provides a function to update the availability of advanced booked cabs based on the current day, marking them as available or cancelling the booking if the requested day has passed.

10. It includes necessary header files for dependencies, such as login, map, admin, clock, cabfinder, and billing.

Overall, the `advancedbooking.h` module handles the advanced booking functionality, including inputting advanced ride details, finding available cabs, calculating distances, updating ride details, generating ride IDs, managing cab availability, and generating advanced booking bills.

5.7 Billing Module

1. It defines the base fare and per kilometre fare as constants using pre-processor directives (`#define`).

2. It provides a function `calculateFare` that takes the distance as input and calculates the fare based on the base fare and per kilometre fare.

3. The `calculateFare` function initializes the price variable with the base fare and calculates the distance fare by multiplying the distance with the per kilometre fare.

4. The calculated distance fare is added to the price, resulting in the total fare.

5. The `calculateFare` function returns the calculated fare as a floating-point value.

Overall, the `billing.h` module provides a simple fare calculation mechanism based on the distance travelled, using predefined base fare and per kilometre fare constants.

5.8 Cab Finder Module

1. The function `dijkstra` is defined to perform Dijkstra's algorithm on a given graph represented by a 2D array to calculate the shortest distances from a source location.

2. The `dijkstra` function uses an array `dist` to store the shortest distances from the source location to all other locations.

3. The function `getNearestCab` is defined to find the nearest available cab based on the customer's location and preferred cab type.

4. The function iterates through the available cabs, checks if they match the preferred cab type and are not advanced booked.

5. For each eligible cab, it finds its location index and compares the distance from the customer location with the current minimum distance and rating.

6. It updates the nearest cab index and tracks the minimum distance and maximum rating accordingly.

7. After iterating through all cabs, it sets the `ride_distance` variable to the minimum distance found for the nearest cab.

8. Finally, the function returns the index of the nearest cab, or -1 if no suitable cab is found.

In summary, this module provides functionality to find the nearest available cab based on the customer's location and preferred cab type using Dijkstra's algorithm for calculating shortest distances.

5.9 Dijkstra's Algorithm

Dijkstra's algorithm is a graph traversal algorithm used to find the shortest path between a source vertex and all other vertices in a weighted graph.

1. Initialize: Set the distance of the source vertex to 0 and the distance of all other vertices to infinity. Mark all vertices as unvisited.

2. Select the vertex with the smallest distance: Among the unvisited vertices, choose the vertex with the smallest distance as the current vertex and mark it as visited.

3. Update distances: For each neighbour of the current vertex that is unvisited, calculate the distance from the source vertex to that neighbour through the current vertex. If this distance is smaller than the previously recorded distance, update the distance.

4. Repeat: Repeat steps 2 and 3 until all vertices have been visited or the destination vertex has been reached.

5. Shortest path: After visiting all vertices or reaching the destination vertex, the algorithm has calculated the shortest path from the source vertex to all other vertices in the graph.

6. Termination: The algorithm terminates when all vertices have been visited, and the shortest distances from the source vertex to all other vertices have been determined.

7. Path reconstruction: If needed, the shortest path from the source vertex to a specific destination vertex can be reconstructed by backtracking from the destination vertex using the recorded distances.

Dijkstra's algorithm guarantees finding the shortest path in a graph with non-negative edge weights. It is widely used in various applications, such as route planning, network routing, and transportation logistics.

5.9.1 Implementation of Dijkstra's Algorithm

The code implements Dijkstra's algorithm to calculate the shortest distances from a source vertex to all other vertices in a graph.

1. The function `dijkstra` takes three parameters: `graph`, which represents the weighted adjacency matrix of the graph, `src`, which is the source vertex from which distances are calculated, and `dist`, an array to store the shortest distances from the source vertex.

2. An array `visited` is initialized to keep track of visited vertices. Initially, all vertices are marked as unvisited.

3. The `dist` array is initialized with a value of `INT_MAX` (infinity) for all vertices except the source vertex `src`, which is set to 0.

4. The main loop runs `MAX_LOCATIONS - 1` times, as it guarantees visiting all vertices except the source vertex.

5. Within each iteration of the loop, the vertex with the minimum distance is selected. This vertex is chosen from the unvisited vertices by comparing their distances in the `dist` array.

6. The selected vertex is marked as visited by setting `visited[u] = 1`.

7. For each unvisited neighbour `v` of the selected vertex `u`, the code checks if the distance from the source vertex to `v` through `u` is smaller than the current recorded distance in the `dist` array. If so, it updates the `dist` array with the new smaller distance.

8. After the main loop finishes, the `dist` array will contain the shortest distances from the source vertex `src` to all other vertices in the graph.

5.10 Get Nearest cab

The `getNearestCab` function finds the nearest available cab based on the customer's location and preferred cab type.

1. Initialize variables: Initialize `dist` array to store the shortest distances from the customer location. Set `minDist` to the maximum integer value, `maxRating` to 0.0, and `nearestCabIndex` to -1.

2. Calculate shortest distances: Use Dijkstra's algorithm (`dijkstra` function) to calculate the shortest distances from the customer location to all other locations in the map. Store the distances in the `dist` array.

3. Iterate through available cabs: Iterate over the available cabs (indexed by `i`) and check if the cab is not advanced booked (`isAdvancedBooked` flag is false) and the cab's type matches the preferred cab type.

4.Find cab's location index: Find the index of the cab's current location in the `Locations` array.

5. Check cab's proximity and rating: If the cab's location index is valid and the distance to the cab is less than `minDist` and the cab's rating is higher than `maxRating`, update `minDist`, `maxRating`, and `nearestCabIndex` accordingly.

6. Update ride distance: Set the global variable `ride_distance` to the calculated `minDist`, which represents the distance between the customer and the nearest cab.

7. Return the nearest cab index: Return the `nearestCabIndex`, which corresponds to the index of the nearest available cab in the `allCabs` array. If no suitable cab is found, the function returns -1.

5.11 Cancellation

The `cancellation.h` module can be explained with the following key points:

1. It defines two arrays, `allRides` and `allAdvancedRides`, to store information about regular and advanced rides, respectively.

2. The variables `numRides` and `numAdvancedRides` keep track of the number of rides stored in each array.

3. The module provides functions to cancel rides based on the provided ride ID. The `cancelRide` function searches for the ride ID in the `allRides` array and removes the

corresponding ride if found. Similarly, the `cancelAdvancedRide` function performs the same operation on the `allAdvancedRides` array.

4. Additionally, the module includes functions to append new rides to the respective arrays, `appendRide` and `appendAdvancedRide`, while checking the maximum limit of rides allowed.

In summary, this module facilitates the cancellation of regular and advanced rides by providing functions to search and remove rides based on their unique ride IDs.

5.12 Clock

1. It includes necessary header files such as "login.h", "map.h", "admin.h", and "booking.h" to access required functions and data structures.

2. The module defines two global variables: `Hour` and `Day`. These variables represent the current hour and day in the system.

3. The `TimeManager` function is responsible for managing the time in the system. It checks if the current hour exceeds 24 and adjusts the day and hour accordingly.

4. If the hour exceeds 24, the function increments the day by the number of days in excess of 24 and resets the hour to the remaining hours.

5. This module ensures that the time in the system is properly managed and updated, allowing for the tracking of days and hours within the application.

In summary, this module provides a mechanism to manage the time within the system, ensuring that the hour and day values are adjusted correctly when the hour exceeds 24. This is crucial for maintaining accurate time-related operations and calculations in the application.

5.13 Map

The map.h module contains data related to the map and locations in the system.

1.Location Information: The module defines an array Locations that stores the names of various locations. Each location is represented as a string.

2.Distance Matrix: The module includes a 2D array Map that represents the distances between different locations. The values in the array represent the distance between two locations. For example, Map[0][1] represents the distance between Chennai and Bangalore.

3.Surge Timing Matrix: The module also includes an adjacency matrix Surge that represents the surge timings while travelling between two locations. Each entry in the matrix indicates the surge hour for a specific route. For example, Surge[0][1] represents the surge hour when travelling from Chennai to Bangalore.

5.14 Interface

This is the main module of the Cab Hailing Application. Here's an explanation of the module:

1.User Authentication: The module allows users to sign up or login. It verifies user credentials and provides access to the admin portal or user portal based on the login status.

2.Admin Portal: If the user is an admin, they are granted access to the admin portal. The admin can create a cab network by generating cab systems. The system's time is updated accordingly.

3.User Portal: If the user is not an admin, they are directed to the user portal. Here, users can book a ride, provide ride details, find the nearest available cab, allocate a cab, calculate the fare, and rate the driver. The system's time is updated accordingly.

***4.Advanced Booking*:** Users can also make advanced bookings by providing additional details such as pickup time and date.

5.Ride Cancellation: Users can cancel their bookings or advanced bookings. The module provides options for cancelling bookings or advanced bookings based on user input. The system's time is updated accordingly.

6.Exit: Users can choose to exit the application, which terminates the program.

Overall, this module serves as the central control for user interactions, booking rides, managing the cab network, and handling cancellations within the Cab Hailing Application.

6. Implementation

We have used **arrays** and **structures** in the project.

Rationale behind this design choice:

Arrays enable fast access to individual elements through their index, making random access operations highly efficient. Locations are stored in an array, and a two-dimensional array is used to store distances between locations; this has allowed easy iteration through elements, facilitating straightforward operations on all elements in an efficient manner. The fixed size of arrays is advantageous here because a constant amount of data needs to be stored, ensuring predictability and control over memory usage avoiding errors such as segmentation fault (core dumped).

Structures provide the ability to group multiple data elements of potentially different types into a single entity, enabling the creation of custom data types that consolidate related information together. For example, we created a structure called cab whose data members are licence id, default location, current location and so on. This promotes better code organization by grouping related data into a single unit, enhancing readability and maintainability. Additionally, structures contribute to modular design by encapsulating data and providing a higher level of abstraction, fostering separation of concerns and promoting a more organized and scalable codebase. The combination of arrays and structures in this project has proved to be a versatile and efficient way of handling and storing data.

User interface design and windows API:

For user interface design we have made use of the **Windows API**:

For Console Text Colour and Border Formatting:

The function setConsoleColor(int colour, int border_color) is used to set the text colour and border formatting for the console. It takes two integer parameters colour and border_color, which represent the desired text colour and border colour, respectively. On Windows, the function SetConsoleTextAttribute() from the Windows API is used to set the text colour.

For border Formatting:

In this project, we have implemented border formatting by repeatedly printing a character (e.g., a hyphen or any other character) a specified number of times. This sequence of characters creates a visual border-like appearance around certain sections of the console output.

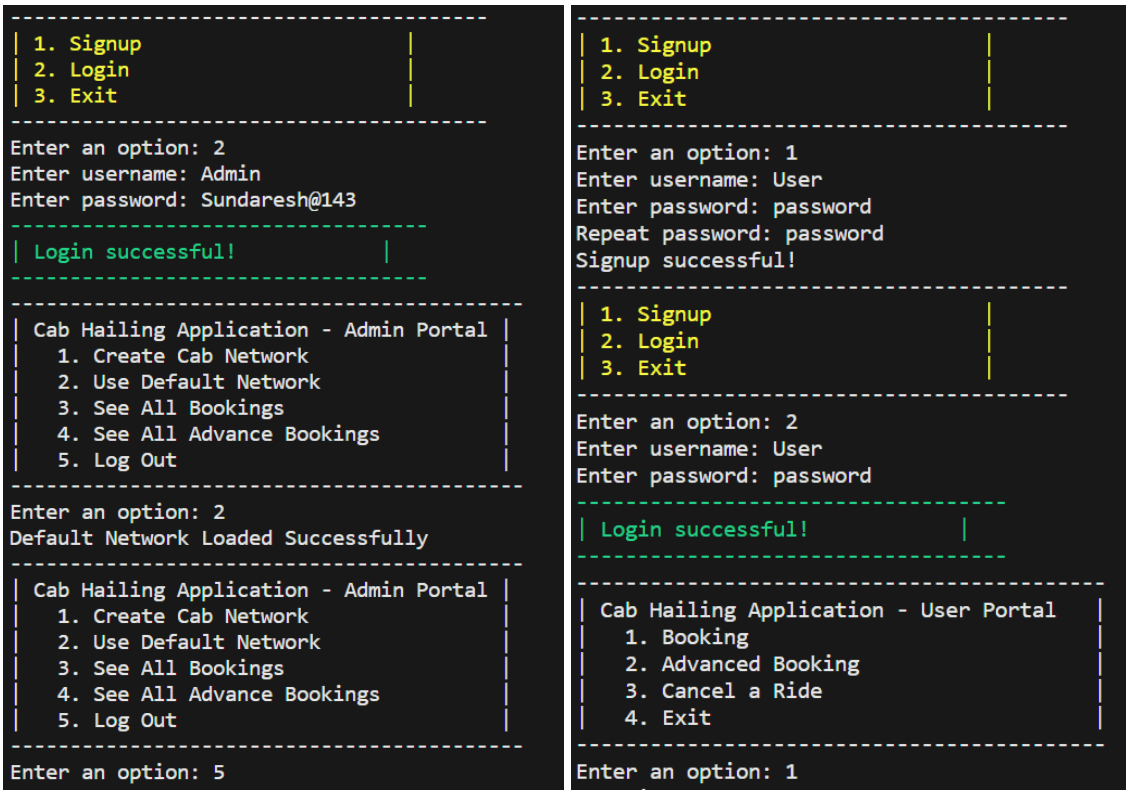
Explanation of Windows API:

We used replit so that all three of us could work on the code simultaneously and the changes would be reflected for all three of us.

7. Test Cases

1. **Basic Test Cases:**

    a. **Booking:**



```
Locations:
1. Chennai
2. Bangalore
3. Goa
4. Yelagiri
5. Mysore
6. Udupi
7. Pondicherry
8. Mangalore
9. Mandya
10. Kannur
11. Murudeshwar
12. Tirunelveli
13. Kundapur
14. Chikkamagaluru
15. Bandipur
16. Madurai
17. Coimbatore
18. Pune
19. Trichy
20. Ernakulam
21. Thanjavur
22. Mumbai
Enter Customer Location: 1
```

```
Locations:
1. Chennai
2. Bangalore
3. Goa
4. Yelagiri
5. Mysore
6. Udupi
7. Pondicherry
8. Mangalore
9. Mandya
10. Kannur
11. Murudeshwar
12. Tirunelveli
13. Kundapur
14. Chikkamagaluru
15. Bandipur
16. Madurai
17. Coimbatore
18. Pune
19. Trichy
20. Ernakulam
21. Thanjavur
22. Mumbai
Enter Drop Location: 3
Cab Types:
1. Toyota
2. Hyundai
3. Nissan
Enter Preferred Cab Type: 2
```

```
Ride Details:
Customer Location: Chennai
Drop Location: Goa
Preferred Cab Type: Hyundai

Cab Type: Hyundai
License ID: L03MBSSQ
Current Location: Goa
Distance from Customer: 899

Ride appended to the array.
The Distance between the Customer Location and the Drop Location is 899

Customer location: Chennai
Drop location: Goa
Driver Rating: 4.50
Ride ID: 5ZY4HSUI
Distance: 899 KM
Fare: Rs. 12236.50
Surge Fare: Rs. 0.00
Total Fare: Rs. 12236.50
Thank you for booking a ride with us!! User

Please rate your driver (from 1 to 5): 5
-----------------------------------------
| Cab Hailing Application - User Portal   |
|   1. Booking                            |
|   2. Advanced Booking                   |
|   3. Cancel a Ride                      |
|   4. Exit                               |
-----------------------------------------
Enter an option: 4
Exiting...
```

b. Advanced Booking:

```
-----------------------------------------
| Cab Hailing Application - User Portal |
|   1. Booking                          |
|   2. Advanced Booking                 |
|   3. Cancel a Ride                    |
|   4. Exit                             |
-----------------------------------------
Enter an option: 2
Locations:
1. Chennai
2. Bangalore
3. Goa
4. Yelagiri
5. Mysore
6. Udupi
7. Pondicherry
8. Mangalore
9. Mandya
10. Kannur
11. Murudeshwar
12. Tirunelveli
13. Kundapur
14. Chikkamagaluru
15. Bandipur
16. Madurai
17. Coimbatore
18. Pune
19. Trichy
20. Ernakulam
21. Thanjavur
22. Mumbai
Enter Customer Location: 1
```

```
Locations:
1. Chennai
2. Bangalore
3. Goa
4. Yelagiri
5. Mysore
6. Udupi
7. Pondicherry
8. Mangalore
9. Mandya
10. Kannur
11. Murudeshwar
12. Tirunelveli
13. Kundapur
14. Chikkamagaluru
15. Bandipur
16. Madurai
17. Coimbatore
18. Pune
19. Trichy
20. Ernakulam
21. Thanjavur
22. Mumbai
Enter Drop Location: 2
Cab Types:
1. Toyota
2. Hyundai
3. Nissan
Enter Preferred Cab Type: 2
Current Time: Day = 1 and Hour = 11
Enter the Day: 2
Enter the Hour: 11
```

```
Ride Details:
Customer Location: Chennai
Drop Location: Bangalore
Preferred Cab Type: Hyundai
Requested day: 2
Requested hour: 11

Cab Type: Hyundai
License ID: L03MBSSQ
Current Location: Goa
Distance from Customer: 899

Ride appended to the array.
The Distance between the Customer Location and the Drop Location is 337

Customer location: Chennai
Drop location: Bangalore
Driver Rating: 4.50
Ride ID: 5ZY4HSUI
Distance: 337 KM
Booking Day: 2
Booking Hour: 11
Fare: Rs. 4649.50
Surge Fare: Rs. 464.95
Total Fare: Rs. 5114.45
Thank you for booking a ride with us, User!

Please rate your driver (from 1 to 5): 5
```

c. Cancellation:

```
-------------------------------------------
| Cab Hailing Application - User Portal   |
|    1. Booking                           |
|    2. Advanced Booking                  |
|    3. Cancel a Ride                     |
|    4. Exit                              |
-------------------------------------------
Enter an option: 3

| Cab Hailing Application - User Cancellation Portal |
|    1. Booking Cancellation                         |
|    2. Advanced Booking Cancellation                |
|    3. Exit                                         |
-----------------------------------------------------
Enter an option: 2
Enter the ID of the Ride you want to cancel: 5ZY4HSUI
Ride with ID 5ZY4HSUI cancelled.
The Cancellation Fares are 33.00
```

2. Handling: Error

a. Invalid credentials:

```
-------------------------------------
| 1. Signup                         |
| 2. Login                          |
| 3. Exit                           |
-------------------------------------
Enter an option: 2
Enter username: Admin
Enter password: Incorrect
-------------------------------------------
| Login failed. Invalid username or password. |
-------------------------------------------
```

```
-------------------------------------
| 1. Signup                         |
| 2. Login                          |
| 3. Exit                           |
-------------------------------------
Enter an option: 2
Enter username: Unknown
Enter password: Random
-------------------------------------------
| Login failed. Invalid username or password. |
-------------------------------------------
```

```
-------------------------------------
| 1. Signup                         |
| 2. Login                          |
| 3. Exit                           |
-------------------------------------
Enter an option: 1
Enter username: User
Username already taken. Please choose a different username.
```

```
-------------------------------------
| 1. Signup                         |
| 2. Login                          |
| 3. Exit                           |
-------------------------------------
Enter an option: 1
Enter username: User2
Enter password: password1
Repeat password: password2
Passwords do not match. Signup failed.
```

b. Entering an invalid location while booking:

```
Locations:
1. Chennai
2. Bangalore
3. Goa
4. Yelagiri
5. Mysore
6. Udupi
7. Pondicherry
8. Mangalore
9. Mandya
10. Kannur
11. Murudeshwar
12. Tirunelveli
13. Kundapur
14. Chikkamagaluru
15. Bandipur
16. Madurai
17. Coimbatore
18. Pune
19. Trichy
20. Ernakulam
21. Thanjavur
22. Mumbai
Enter Customer Location: 24
Please Enter a Valid Location
```

```
Locations:
1. Chennai
2. Bangalore
3. Goa
4. Yelagiri
5. Mysore
6. Udupi
7. Pondicherry
8. Mangalore
9. Mandya
10. Kannur
11. Murudeshwar
12. Tirunelveli
13. Kundapur
14. Chikkamagaluru
15. Bandipur
16. Madurai
17. Coimbatore
18. Pune
19. Trichy
20. Ernakulam
21. Thanjavur
22. Mumbai
Enter Drop Location: 0
Please Enter a Valid Location
```

```
Enter Customer Location: 1
Locations:
1. Chennai
2. Bangalore
3. Goa
4. Yelagiri
5. Mysore
6. Udupi
7. Pondicherry
8. Mangalore
9. Mandya
10. Kannur
11. Murudeshwar
12. Tirunelveli
13. Kundapur
14. Chikkamagaluru
15. Bandipur
16. Madurai
17. Coimbatore
18. Pune
19. Trichy
20. Ernakulam
21. Thanjavur
22. Mumbai
Enter Drop Location: 1
The Pickup Location and the Drop Location are the same
```

```
Cab Types:
1. Toyota
2. Hyundai
3. Nissan
Enter Preferred Cab Type: 4
Please Enter a Valid Cab Type
```

c. **Invalid time entered while advanced Booking:**

```
Current Time: Day = 1 and Hour = 11
Enter the Day: 1
Enter the Hour: 10
Enter a valid input.
Enter the Hour: 2
Enter a valid input.
```

d. **Absence of Cab Network:**

```
----------------------------------------
| 1. Signup                            |
| 2. Login                             |
| 3. Exit                              |
----------------------------------------
Enter an option: 1
Enter username: User
Enter password: password
Repeat password: password
Signup successful!
----------------------------------------
| 1. Signup                            |
| 2. Login                             |
| 3. Exit                              |
----------------------------------------
Enter an option: 2
Enter username: User
Enter password: password
----------------------------------
| Login successful!              |
----------------------------------
No Cab Network has been Created
----------------------------------------
| 1. Signup                            |
| 2. Login                             |
| 3. Exit                              |
----------------------------------------
Enter an option:
```

e. Invalid ride ID entered at the time of Cancellation:

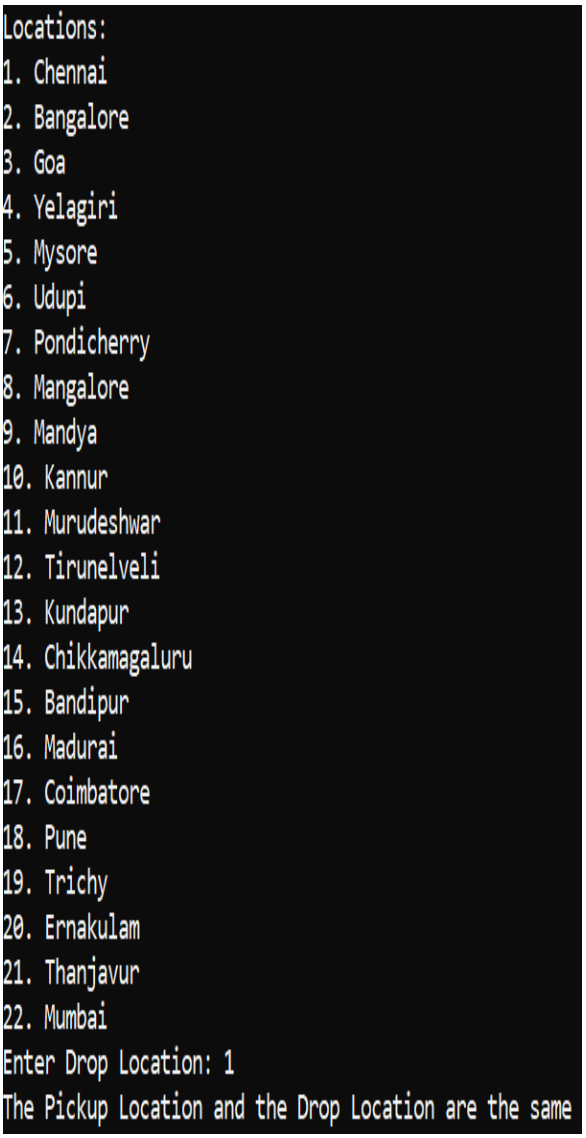
```
Customer location: Chennai
Drop location: Bangalore
Driver Rating: 4.50
Ride ID: 5ZY4HSUI
Distance: 337 KM
Fare: Rs. 4649.50
Surge Fare: Rs. 464.95
Total Fare: Rs. 5114.45
Thank you for booking a ride with us!! User
```

```
| Cab Hailing Application - User Cancellation Portal |
|   1. Booking Cancellation                          |
|   2. Advanced Booking Cancellation                 |
|   3. Exit                                          |
----------------------------------------
Enter an option: 1
Enter the ID of the Ride you want to cancel: 5ZY4HSKJ
Ride with ID 5ZY4HSKJ not found.
```

```
| Cab Hailing Application - User Cancellation Portal |
|   1. Booking Cancellation                          |
|   2. Advanced Booking Cancellation                 |
|   3. Exit                                          |
----------------------------------------
Enter an option: 1
Enter the ID of the Ride you want to cancel: 5ZY4HSUI
Ride with ID 5ZY4HSUI cancelled.
The Cancellation Fares are 33.00
```

f. Same pick up and drop location entered:


| Cab Hailing Application - User Portal |
| 1. Booking |
| 2. Advanced Booking |
| 3. Cancel a Ride |
| 4. Exit |
Enter an option: 1
Locations:
1. Chennai
2. Bangalore
3. Goa
4. Yelagiri
5. Mysore
6. Udupi
7. Pondicherry
8. Mangalore
9. Mandya
10. Kannur
11. Murudeshwar
12. Tirunelveli
13. Kundapur
14. Chikkamagaluru
15. Bandipur
16. Madurai
17. Coimbatore
18. Pune
19. Trichy
20. Ernakulam
21. Thanjavur
22. Mumbai
Enter Customer Location: 1


Locations:
1. Chennai
2. Bangalore
3. Goa
4. Yelagiri
5. Mysore
6. Udupi
7. Pondicherry
8. Mangalore
9. Mandya
10. Kannur
11. Murudeshwar
12. Tirunelveli
13. Kundapur
14. Chikkamagaluru
15. Bandipur
16. Madurai
17. Coimbatore
18. Pune
19. Trichy
20. Ernakulam
21. Thanjavur
22. Mumbai
Enter Drop Location: 1
The Pickup Location and the Drop Location are the same

8. Limitations

It is essential to consider these limitations when evaluating the functionality and usability of the cab hailing system.

8.1. Limited Set of Destinations and Pick-Up Points:

The current implementation of the system requires users to choose from a predefined set of destinations and pick-up points. It does not allow for flexibility in selecting alternative or custom locations. Users are constrained by the available options provided by the system.

8.2. Inflexibility in Changing Destinations:

Once a destination is selected and a ride is booked, it is not feasible to change the destination at any point in time. Users must ensure that the correct destination is specified during the booking process, as modifications or updates to the destination are not supported by the system.

8.3. Single System Operation:

The cab hailing system is designed to operate on a single system or platform. It does not have the capability to support multiple users simultaneously logging in and accessing the system

concurrently. This limitation restricts the scalability of the system, particularly in scenarios where a large number of users may need to use the application concurrently.

Future enhancements could focus on addressing these constraints to provide a more flexible and scalable solution for users, allowing them to choose from a wider range of destinations and pick-up points, accommodate changes in the destination, and enable multi-user access for improved usability and efficiency.

9. Observations from Social, Ethical and Legal perspective

During the development of any application, it is crucial to consider the social, ethical, and legal aspects to ensure its successful implementation on a large scale. We recognize and acknowledge the significance of addressing these aspects to ensure the application's responsible and sustainable deployment. The observations we have made in these areas, are as follows:

9.1 Observations Pertaining to Social Perspective:

Observations Pertaining to Social Perspective:

1. Impact on Travel Convenience: Cab hailing applications have transformed the way people travel by providing a convenient and on-demand transportation service. Users can easily book rides with a few taps on their smart phones, eliminating the need to wait for public transportation or search for parking spaces.

2. Increased Accessibility: These applications have made transportation more accessible to a wider range of people, including those who may not own private vehicles or have limited access to public transportation. The availability of cabs at various locations and flexible ride options cater to diverse transportation needs.

3. Job Creation: The emergence of cab hailing applications has created employment opportunities in the transportation sector. Drivers can join ride-hailing platforms and earn income by providing transportation services. Additionally, support staff, such as

customer service representatives and technical support personnel, is employed to ensure smooth operations of the platforms.

9.2 Observations Pertaining to Ethical Perspective:

1. Driver Exploitation: There are concerns about the treatment of drivers working for cab hailing platforms. Issues such as low pay rates, long working hours, and lack of job security have been raised. Some drivers may face challenges in earning a sustainable income and maintaining a healthy work-life balance.

3. Fairness and Transparency: The algorithms used by cab hailing applications to assign rides and calculate fares can be a subject of ethical debate. Questions arise about the fairness and transparency of these algorithms, ensuring that customers and drivers are treated equitably without any bias or discrimination.

9.3 Observations Pertaining to Legal Perspective:

1. Regulatory Challenges: The emergence of cab hailing applications has presented new regulatory challenges for governments and transportation authorities. Traditional transportation regulations often do not cover these innovative business models, leading to debates on how to regulate and monitor their operations effectively.

2. Licensing and Permitting: Cab hailing companies and their drivers may be subject to specific licensing and permitting requirements imposed by local transportation authorities. Ensuring compliance with these regulations is essential to maintain the integrity and safety of the transportation industry.

3. Insurance Requirements: The insurance landscape for cab hailing services is complex. There is a need to address gaps in coverage, particularly during different phases of a ride (e.g., waiting for a customer, en route to a destination, or when a passenger is on board). Clarifying insurance requirements and developing appropriate coverage models are essential for protecting both drivers and passengers.

4. Driver Classification: The legal status of drivers working for cab hailing companies is a contentious issue. Some argue that drivers should be classified as employees rather than independent contractors, which would entitle them to benefits and protections offered to traditional employees. This classification has legal implications for labour laws and employment rights.

5. Consumer Protection: Ensuring consumer protection is crucial in the cab hailing industry. Legal frameworks need to address issues such as price transparency, dispute resolution mechanisms, driver background checks, and passenger safety standards.

10. Learning Outcomes

10.1. Understanding Transportation Systems:

Engaging with this cab hailing application problem statement has helped us develop a deeper understanding of transportation systems and the complexities involved in efficiently allocating resources to meet customer demands.

10.2. Software Development Skills:

Through the design and implementation of a comprehensive cab hailing application, it has helped us enhance our software development skills. We have gained hands-on experience in user registration, ride booking, fare calculation, and driver allocation, allowing us to improve our proficiency in working with real-time data, optimizing algorithms, and delivering a seamless user experience.

10.3. Data Management and Privacy:

Developing the cab hailing application has provided us with insights into data management and privacy considerations. We have learned how to handle sensitive user data, such as personal information and payment details, and implement robust security measures to protect user privacy.

10.4. Ethical Awareness:

Exploring the social and ethical aspects of cab hailing applications has raised our awareness of the ethical dilemmas and challenges associated with such platforms. We have developed a deeper understanding of the importance of fair treatment of drivers, user data privacy, and transparency in algorithms. This knowledge allows us to make informed ethical considerations in our future software development projects.

10.5. System Scalability and Performance:

Working on the cab hailing application has exposed us to the challenges of handling a high volume of customer requests and real-time vehicle allocation. We have learned about system scalability, load balancing, and performance optimization techniques, enabling us to address these challenges effectively in our future projects.

These learning outcomes reflect how our engagement with the cab hailing application problem statement has contributed to our understanding and growth in various aspects of software development and the ethical considerations involved in building such applications.

11. References

11.1 Books:

1. *Designing Data-Intensive Applications* by Martin Kleppmann
2. *The Pragmatic Programmer: Your Journey to Mastery* by Andrew Hunt and David Thomas
3. *Let Us C* by Yashwant Kanetkar
4. *C Programming Language* by Brian W. Kernighan and Dennis M. Ritchie

11.2 Websites:

1. https://stackoverflow.com/
2. https://www.geeksforgeeks.org/
3. https://replit.com/

11.3 Tools used:

1. https://www.figma.com/files/recents-and-sharing?fuid=1251137085898410223
2. https://online.visual-paradigm.com/diagrams/features/flowchart-tool/
3. https://www.overleaf.com/learn/latex/Choosing_a_LaTeX_Compiler
4. https://www.onlinegdb.com/