CAB HAILING APPLICATION

UCS2201 – Fundamentals and Practice of Software Development

A PROJECT REPORT

Submitted By

Varshini Venkatesh 3122 22 5001 153

Vidisha Desai 3122 22 5001 154

Vineeth U 3122 22 5001 160

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 " Varshini Venkatesh (3122 22 5001 153), Vidisha Desai (3122 22 5001 154) and Vineeth U (3122 22 5001 160)" 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

1. ABSTRACT

The Cab Hailing Software System streamlines the process of booking, tracking, and managing cab rides. It offers a user-friendly interface for users to input their details, login, and register securely. The system efficiently assigns cabs based on minimum distance and estimated time of arrival. It generates accurate bills for completed rides and incorporates a rating system for users to provide feedback on their experience. The software system aims to enhance user convenience, optimize cab allocation, and improve customer satisfaction.

By utilizing the Cab Hailing Software System, users can easily book cabs and enjoy a seamless experience. The system assigns cabs based on factors like minimum distance and estimated time of arrival, ensuring efficient and timely service. After completing a ride, users receive accurate bills that detail the distance traveled and fare breakdown. Additionally, the rating system allows users to provide feedback, promoting accountability and continuous improvement.

Overall, the Cab Hailing Software System provides a user-friendly solution for booking and managing cab rides. It optimizes the allocation of cabs, generates accurate bills, and encourages user feedback. By adopting this software system, cab service providers can enhance customer satisfaction, improve operational efficiency, and thrive in the competitive cab hailing industry.

2. INTRODUCTION

In this era of technological advancements, the transportation industry has witnessed significant transformations, and cab hailing services have become an integral part of modern commuting. With the introduction of cab hailing software systems, the process of booking, tracking, and managing cab rides has been revolutionized.

Operating a successful cab service involves various intricate tasks, including seamless ride bookings, effective driver management, real-time tracking, and accurate billing. The development of a comprehensive cab hailing software system brings together these essential functions into a unified platform. By automating processes, providing real-time ride tracking, generating precise bills, and offering valuable insights into business performance, the software system aims to enhance operational efficiency, accuracy, and customer service.

Our project's primary objective is to design and develop a user-friendly cab hailing software system that significantly improves the daily workload of cab service providers while elevating the quality of service provided. The subsequent sections of this project will delve into the detailed analysis, design, and implementation aspects, presenting a comprehensive solution to address the challenges and requirements of the cab hailing domain.

3. 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:

- The customer should 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.

4. EXTENDED EXPLORATION OF THE PROBLEM STATEMENT

- Considered the perspectives of various stakeholders, including commuters, cab drivers, and the company operating the cab hailing service.
- Conducted research on existing cab hailing applications in the market, such as Uber, Ola and Lyft, to understand their features with respect to each stakeholder:

  a) Commuters: Easy registration and booking process ,Option to choose from various types of cabs, Estimated fare calculation before booking the ride.

b) Drivers : Option to register as a driver, Transparent earnings and incentives structure.Flexible work schedule and the ability to choose ride requests.

c) Company : Platform management for efficient matching of riders and drivers, Marketing and promotions to attract new riders and drivers, Analytics and data insights for optimizing operations and improving efficiency.

- Decided to narrow down to only the commuters' perspective and identified the key components in the project - user registration and authentication, ride request management, driver allocation and dispatching, fare estimation and rating system.

5. FUNCTIONAL AND NON FUNCTIONAL REQUIREMENTS

FUNCTIONAL REQUIREMENTS:

- User Management:
  a) User Registration: Users should be able to register with their personal details.
  b) User Login: Registered users should be able to log in to access their profile and perform actions.
  c) View Profile: Users should be able to view their profile information.
  d) Cab FAQ: Users should be able to view frequently asked questions related to cab services.

- Admin Management:
  a) Admin Login: The admin should be able to log in with a password to access the admin portal.
  b) Manage Drivers: The admin should be able to add, update, and remove driver details.
  c) Manage Cab Types: The admin should be able to add, update, and remove different types of cabs.
  d) Perform Administrative Tasks: The admin should have the ability to perform administrative tasks related to the system.

- Cab Allocation/Bill Generation:
    a) Spot Booking: Users should be able to book cabs for immediate pickup by providing pickup and drop-off locations.
    b) Advance Booking: Users should be able to book cabs in advance by specifying pickup and drop-off locations and requested booking time.
    c) Shortest Path Calculation: The system should calculate the shortest path between pickup and drop-off locations.
    d) Allot Cab: The system should allot a cab to the user based on the chosen cab type and availability.
    e) Cancellation: Users should be able to cancel their cab booking, and appropriate cancellation fees should be applied.
    f) Bill Generation: The system should generate bills for cab bookings, including surge fees, cancellation fees, and the final price.

NON FUNCTIONAL REQUIREMENTS:

- Performance: The system should respond quickly to user actions, providing a seamless user experience.

- Reliability: The system should handle errors gracefully and provide informative error messages to users. User data should be stored securely and protected from unauthorized access or modifications.

- Usability: The user interface should be intuitive and easy to navigate. Error messages and prompts should be clear and user-friendly.

- Security: User passwords should be stored securely using appropriate techniques. The system should implement authentication mechanisms to ensure that only authorized users can access sensitive functionalities.

- Maintainability: The code should be well-structured, modular, and follow coding best practices. Proper documentation should be provided to aid in system maintenance and future enhancements.

6. ANALYSIS USING DATA FLOW DIAGRAMS

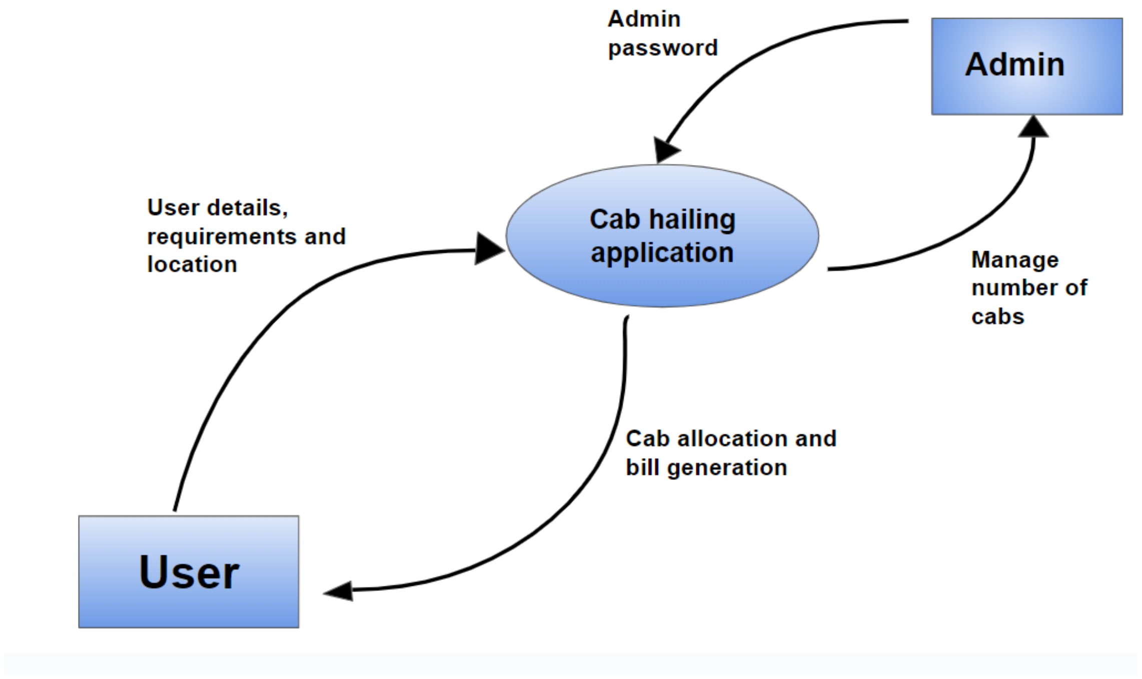

Admin password
Admin
User details, requirements and location
Cab hailing application
Manage number of cabs
Cab allocation and bill generation
User

FIG 1 : CONTEXT DIAGRAM

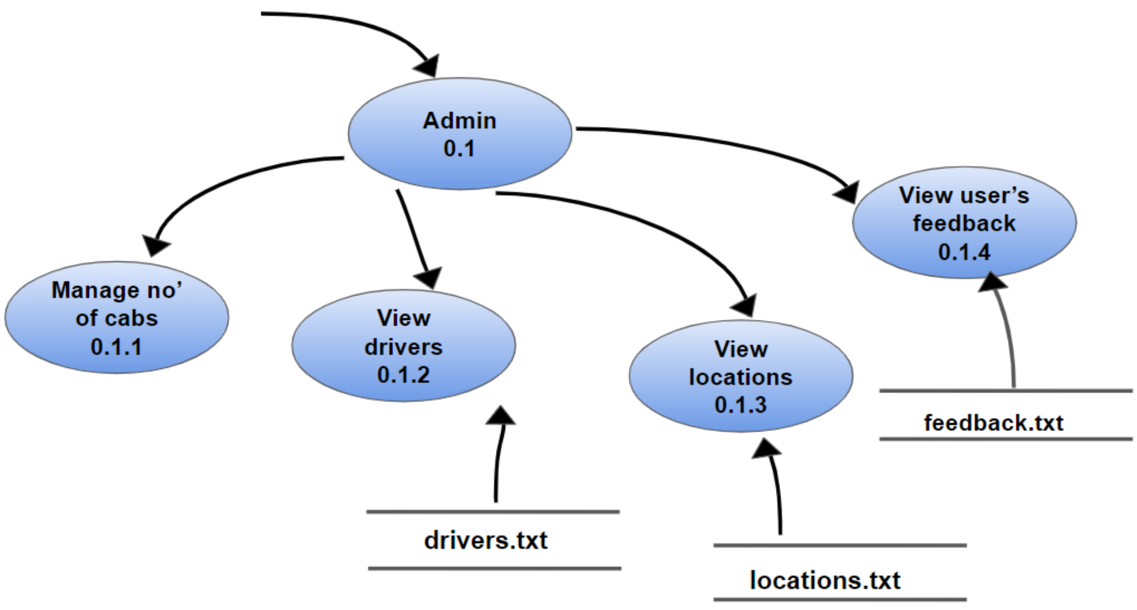


FIG 2 : LEVEL 1 DATA FLOW DIAGRAM

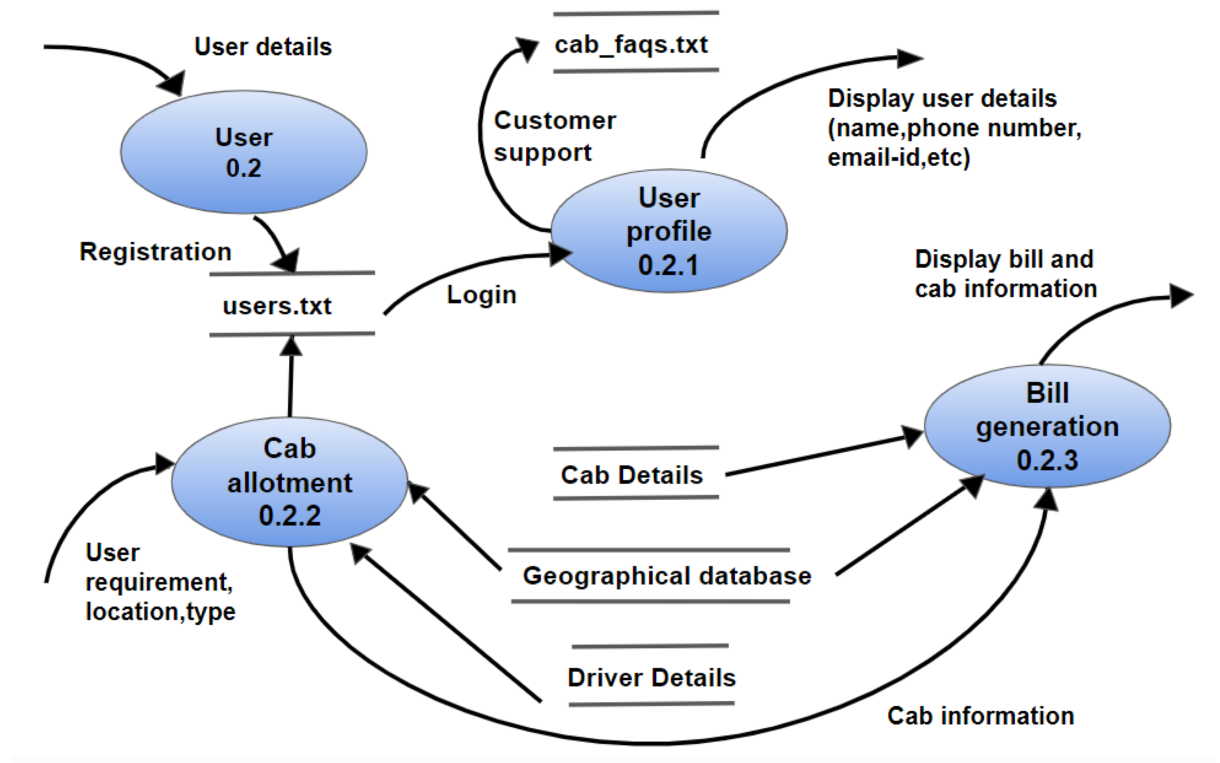
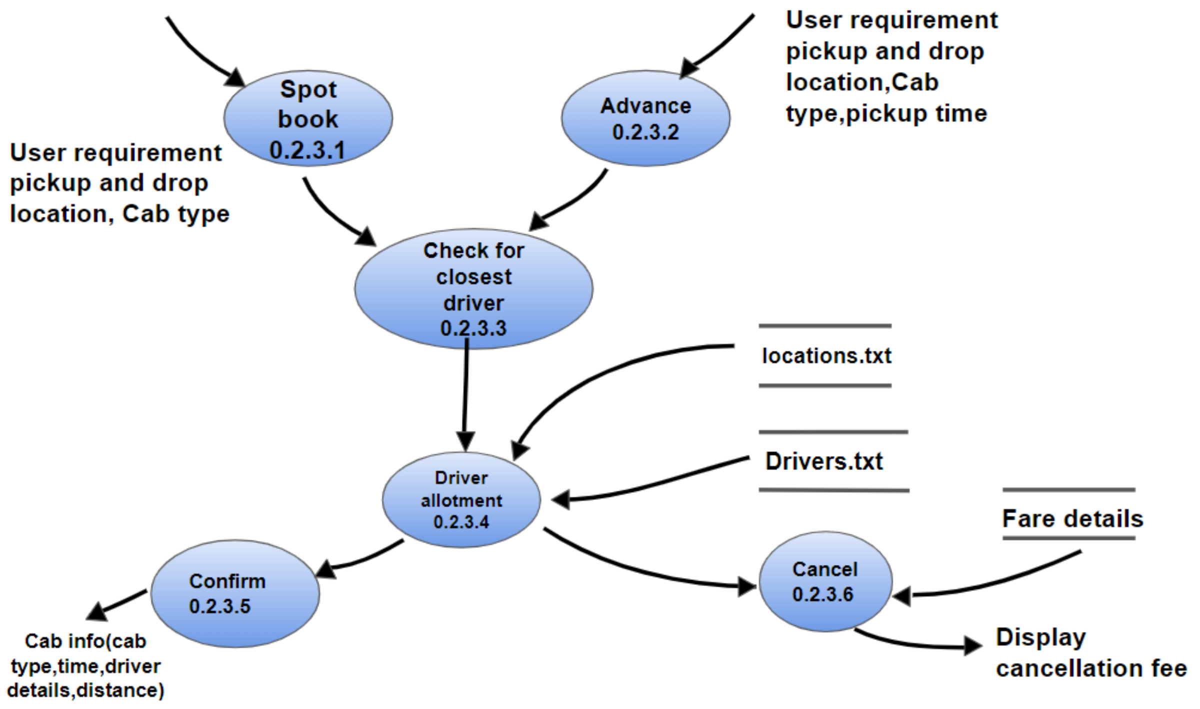


FIG 3 : LEVEL 1 DATA FLOW DIAGRAM



Spot book 0.2.3.1
Advance 0.2.3.2
User requirement pickup and drop location,Cab type,pickup time
User requirement pickup and drop location, Cab type
Check for closest driver 0.2.3.3
locations.txt
Drivers.txt
Driver allotment 0.2.3.4
Fare details
Confirm 0.2.3.5
Cancel 0.2.3.6
Cab info(cab type,time,driver details,distance)
Display cancellation fee

FIG 4 : LEVEL 2 DATA FLOW DIAGRAM
(CANCELLATION MODULE)

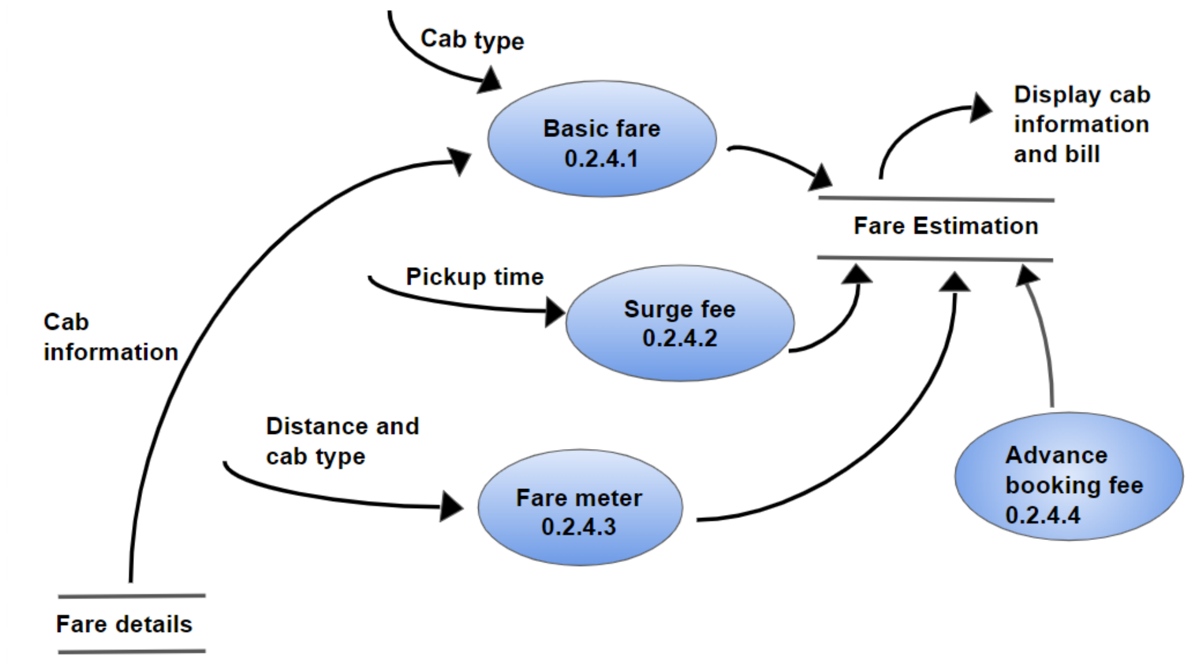


FIG 5 : LEVEL 2 DATA FLOW DIAGRAM (BILL GENERATION MODULE)

7. DETAILED DESIGN

7.1 ARCHITECTURE DIAGRAM

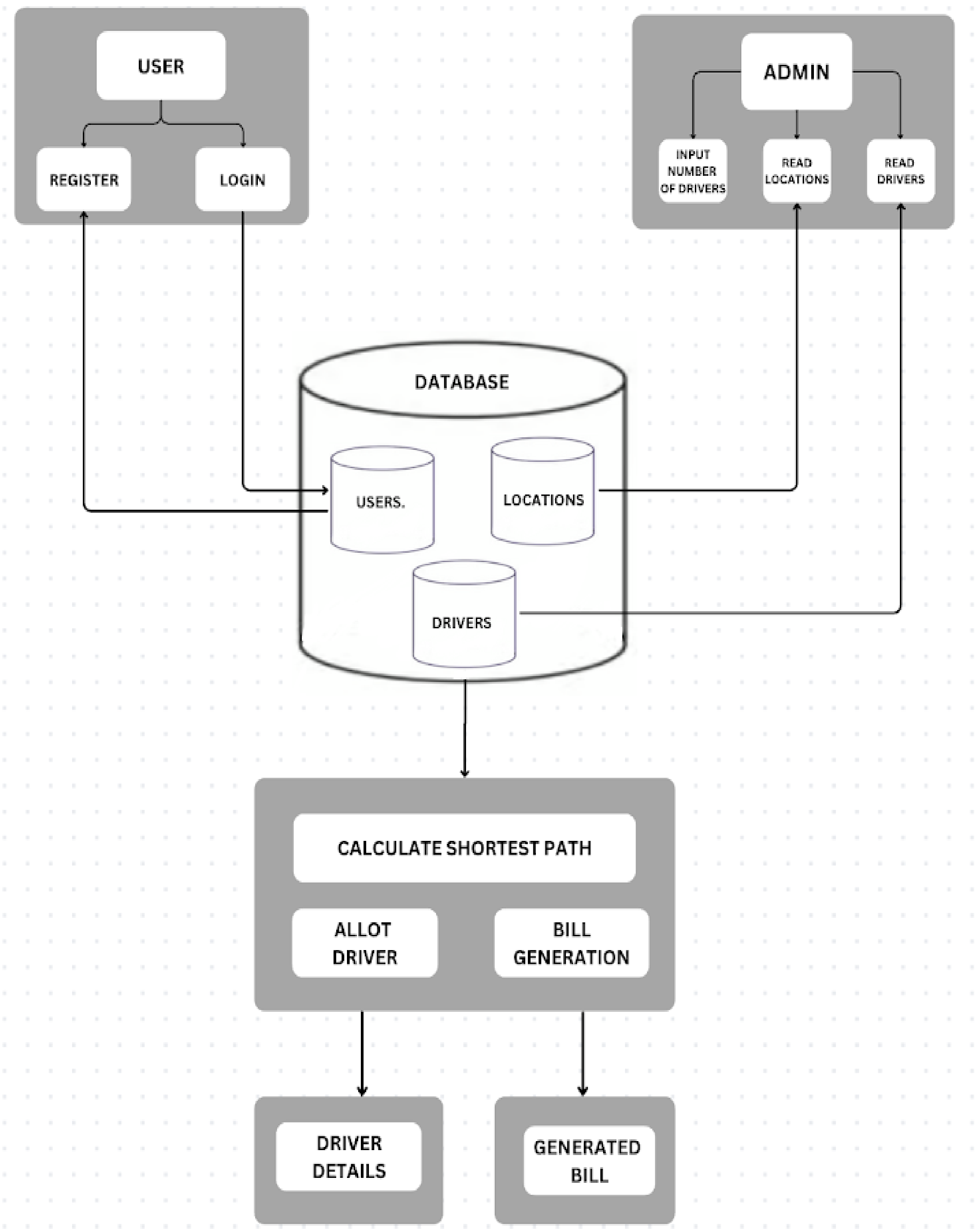


FIG 5 : ARCHITECTURE DIAGRAM

7.2 STRUCTURE DIAGRAMS

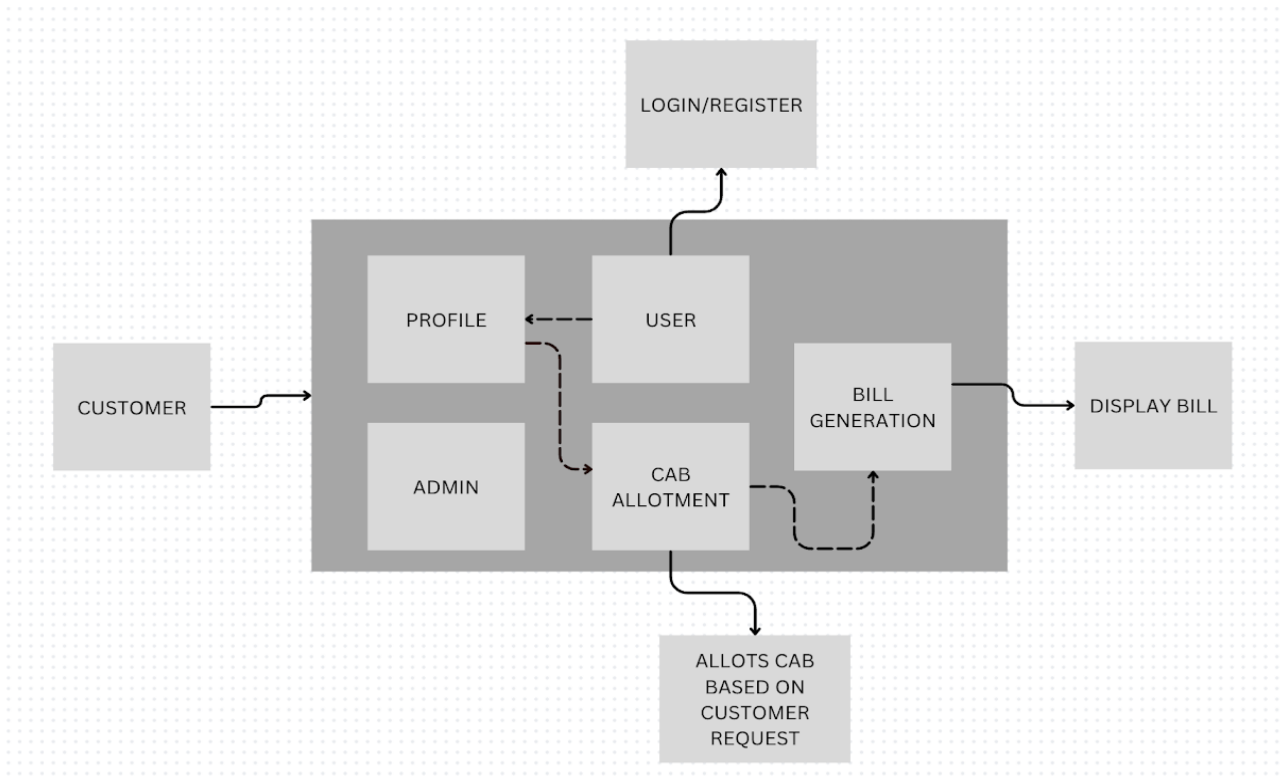


FIG 6 : OVERALL STRUCTURE DIAGRAM

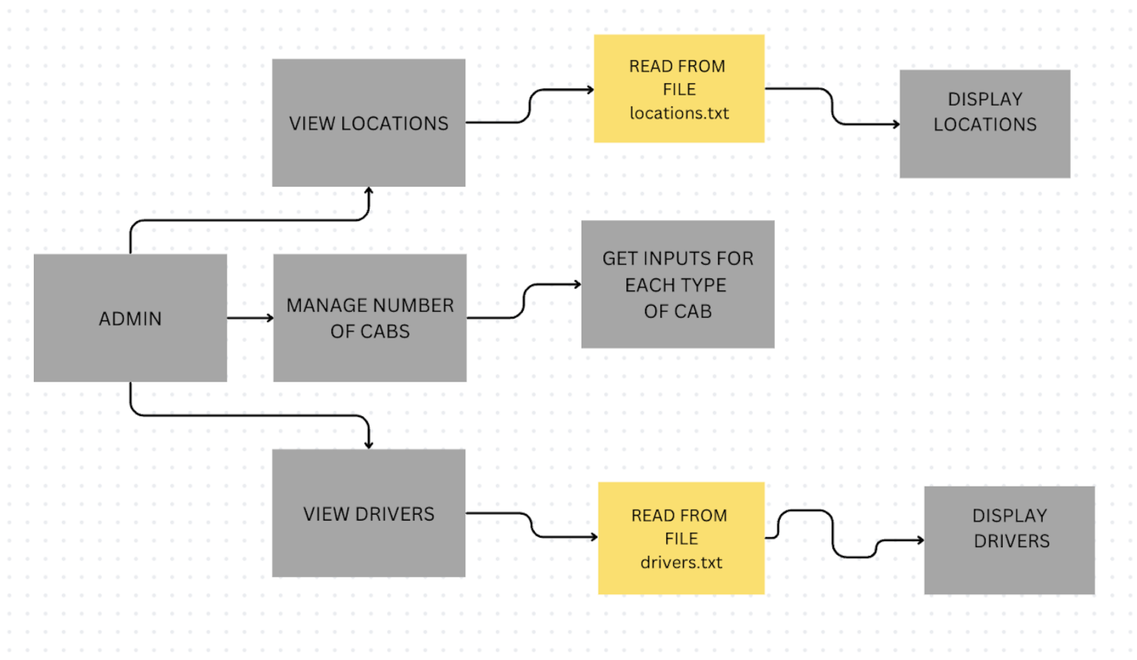

VIEW LOCATIONS
READ FROM FILE locations.txt
DISPLAY LOCATIONS
ADMIN
MANAGE NUMBER OF CABS
GET INPUTS FOR EACH TYPE OF CAB
VIEW DRIVERS
READ FROM FILE drivers.txt
DISPLAY DRIVERS

FIG 7 : STRUCTURE DIAGRAM - ADMIN MODULE

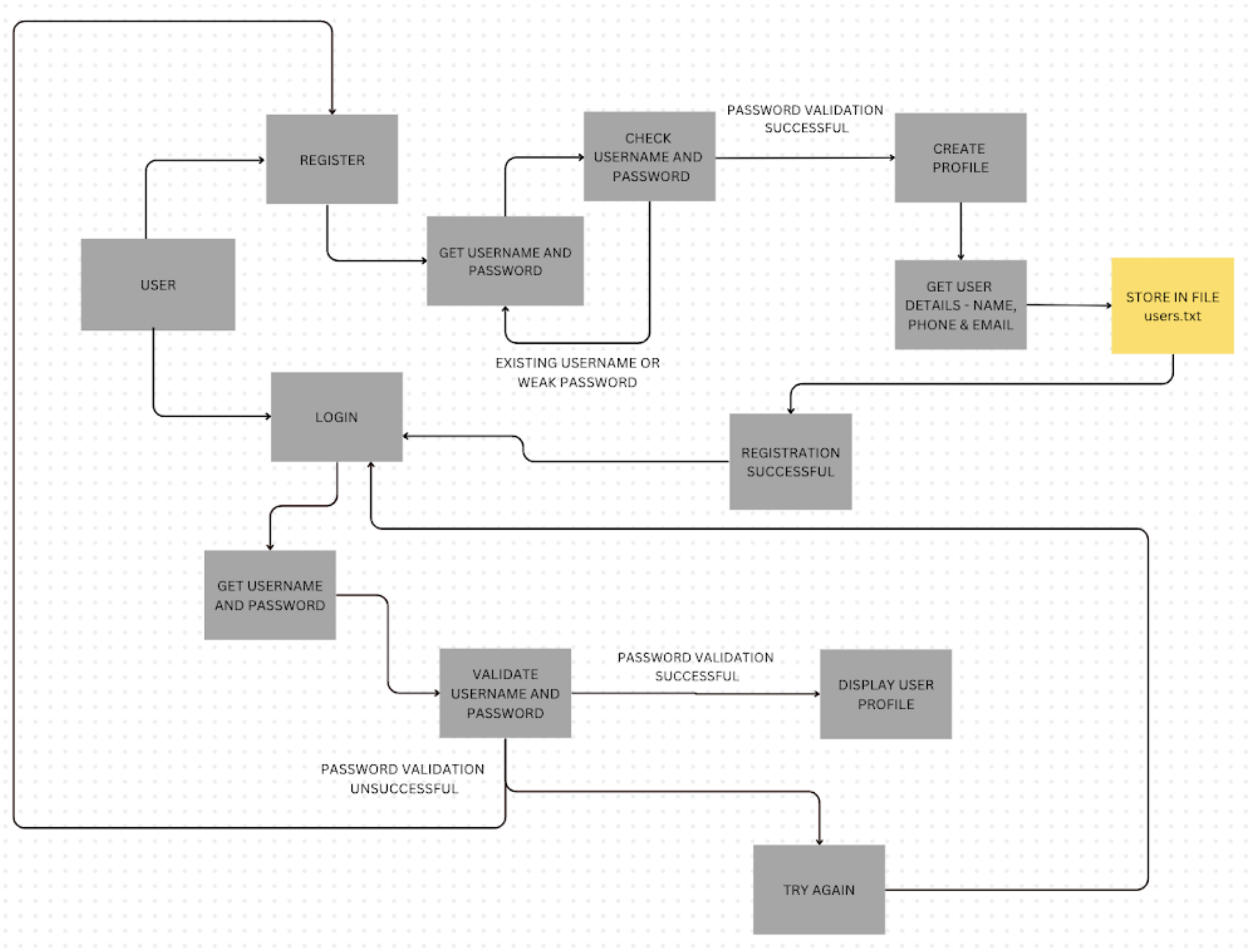

REGISTER
CHECK USERNAME AND PASSWORD
PASSWORD VALIDATION SUCCESSFUL
CREATE PROFILE
USER
GET USERNAME AND PASSWORD
GET USER DETAILS - NAME, PHONE & EMAIL
STORE IN FILE users.txt
EXISTING USERNAME OR WEAK PASSWORD
LOGIN
REGISTRATION SUCCESSFUL
GET USERNAME AND PASSWORD
VALIDATE USERNAME AND PASSWORD
PASSWORD VALIDATION SUCCESSFUL
DISPLAY USER PROFILE
PASSWORD VALIDATION UNSUCCESSFUL
TRY AGAIN

FIG 8 : STRUCTURE DIAGRAM - USER MODULE

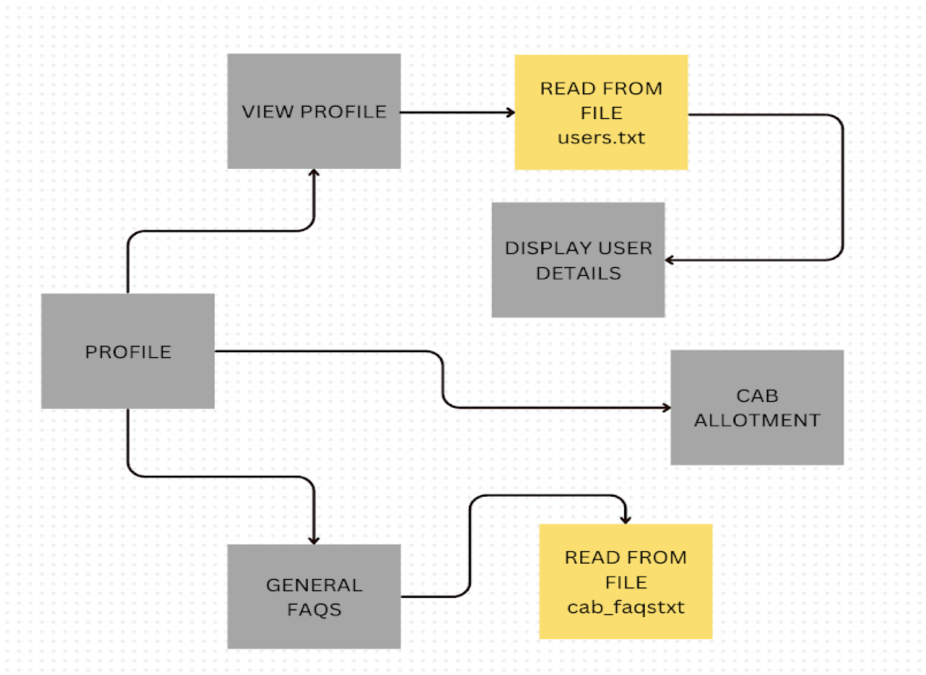


FIG 9 : STRUCTURE DIAGRAM - PROFILE MODULE

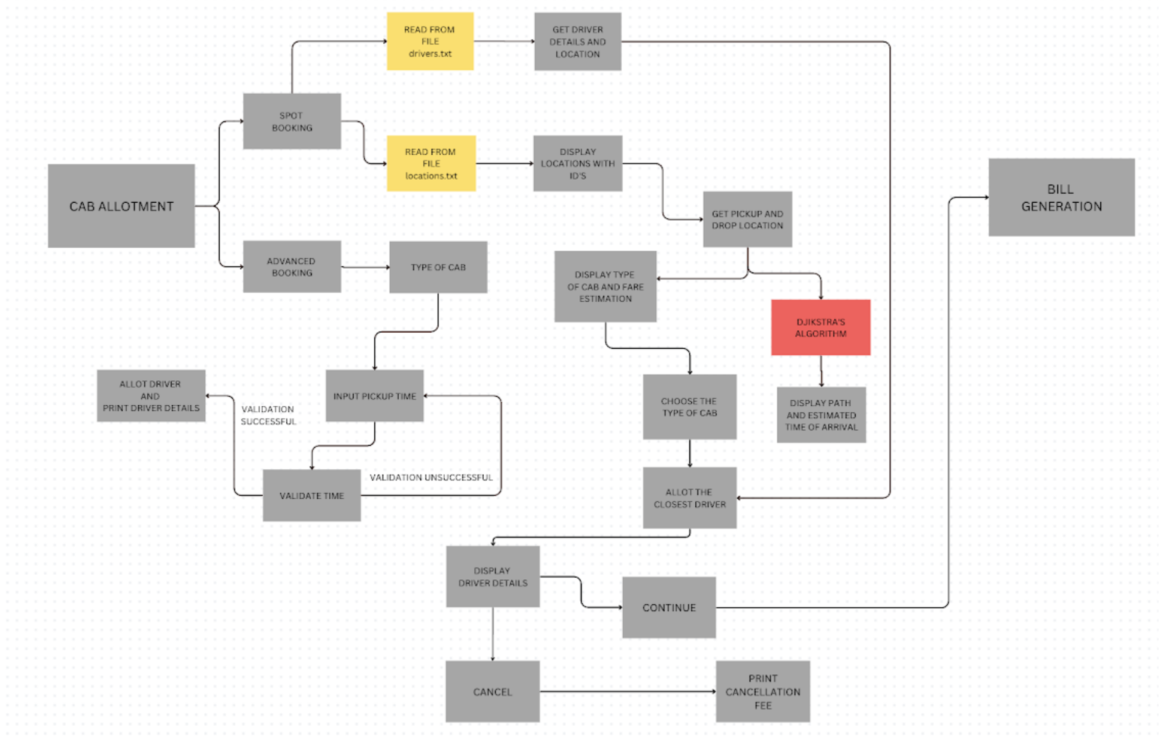


FIG 10 : STRUCTURE DIAGRAM - CAB ALLOTMENT MODULE

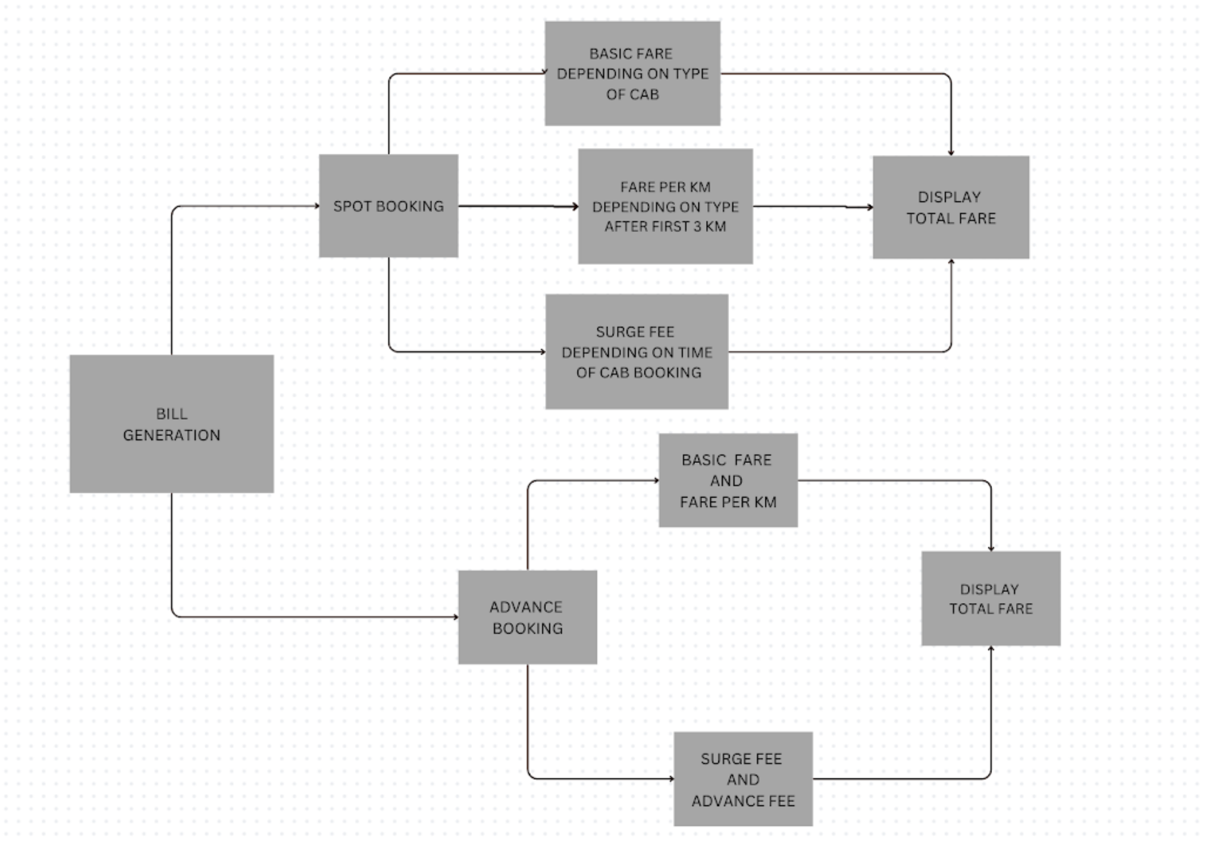

BILL GENERATION
SPOT BOOKING
BASIC FARE DEPENDING ON TYPE OF CAB
FARE PER KM DEPENDING ON TYPE AFTER FIRST 3 KM
SURGE FEE DEPENDING ON TIME OF CAB BOOKING
DISPLAY TOTAL FARE
ADVANCE BOOKING
BASIC FARE AND FARE PER KM
SURGE FEE AND ADVANCE FEE
DISPLAY TOTAL FARE

FIG 11 : STRUCTURE DIAGRAM - BILL GENERATION MODULE

8. EXPLANATION OF EACH MODULE

ADMIN MODULE

The Admin function is responsible for managing various aspects of the system as an administrator. It provides a menu-driven interface to perform different administrative tasks. Here's a brief explanation of each module:

- **Manage number of cabs**: This module allows the administrator to specify the number of available cabs for each type. It prompts the administrator to enter the number of Minis, Sedans, SUVs, Autos, and Bikes.

- **View locations:** This module displays the list of locations available in the system. It retrieves the location data from the `Graph` structure and prints the location IDs and names.

  The **printLocations** function is used to print the list of locations by iterating over the locations stored in the `Graph` structure.

- **View drivers**: This module shows the details of all the drivers in the system. It retrieves the driver data from the Graph structure and prints information such as driver ID, name, car type, phone number, car number, and rating.

  The **printDrivers** function is used to print the details of all the drivers by iterating over the drivers stored in the Graph structure.

USER MODULE

The User module is responsible for managing user-related operations in a cab allotment system. It includes two sub-modules, Register and Login, which handle user registration and authentication processes, respectively.

Register Module
The Register module provides functionalities for user registration. It includes the following functions:

- **void maskPasswordInput(char password[])**: This function securely masks user input for password characters, allowing the user to enter a password without it being displayed on the screen. It utilizes terminal settings to hide the password input.

- **bool isValidEmail(char *email)**: This function validates the format of an email address and returns a boolean value indicating whether it is valid or not. It checks for the presence of the "@" symbol, at least one dot ".", and ensures their appropriate placement.

- **bool isValidPhoneNumber(char *number)**: This function validates the format of a phone number and returns a boolean value indicating its validity. It checks if the phone number consists of exactly 10 digits.

- **int isStrongPassword(char password[]):** This function checks the strength of a given password, ensuring it meets certain criteria such as minimum length (8 characters), containing at least one uppercase letter, one lowercase letter, one number, and one special character.

- **void registerUser()**: This function handles the registration process for a new user. It prompts the user to enter a unique username, validates its availability, and prompts for a strong password. It also collects additional profile details like first name, last name, phone number, and email. The entered details are stored in a file named "users.txt" . The user details are typically stored in a User struct, which stores information such as username, password, first name, last name, phone number, and email.

Login module
The Login sub-module provides functionalities for user authentication and login. It includes the following function:

- **int loginUser():** This function handles the user login process. It prompts the user to enter their username and password, validates the credentials by checking against the entries stored in "users.txt" or a suitable storage mechanism, and allows the user to log in upon successful authentication. The user details, including the User struct, are used for comparison during the login process.

PROFILE MODULE

The Profile module handles various functionalities related to user profiles within the cab allotment system. It includes the following functions:

- **cab_faq()** :The function displays frequently asked questions (FAQs) related to the cab service.
  It opens the "cab_faqs.txt" file and reads each line, printing them on the screen. Once all the FAQs are displayed, the file is closed, and a message indicating redirection to the menu is shown. The function pauses execution for 3 seconds before returning.

- **cab_allotment():** The function represents the entry point for the cab allotment module. It initiates the cab allotment process and handles the allocation of cabs to users.

- **view_profile()**: The function allows users to view their profile details after a successful login. It opens the "users.txt" file containing user details and reads each User struct from the file.
  The function compares the username and password stored in the global variables username_check and password_check with each entry in the file to find a match. If a match is found, the user's name, phone number, and email are displayed on the screen.
  A message indicating redirection to the menu is shown, and the function pauses execution for 3 seconds before returning. If no match is found, indicating that the user's details were not found in the file, a message indicating the user was not found is displayed.

CAB ALLOTMENT MODULE:

The Cab Allotment Module is a crucial component of a cab booking system that handles the task of assigning a driver with the specified cab type to a customer based on their pickup location. It consists of two sub-modules: Spot Booking and Advance Booking.

SPOT BOOKING MODULE

Spot booking refers to the process of booking a cab for immediate use, without any prior reservation. It includes following functions:

- **displayAvailableCabTypes():**The function `displayAvailableCabTypes` lists the available cab types and their corresponding fare rates based on the given distance. It checks the availability of each cab type and calculates the fare rate by adjusting the base fare rate with an additional charge per kilometer beyond the initial distance. The function presents this information to the user in a clear and concise manner.

- **printLocations():** The function printLocations takes a pointer to a Graph structure as input and prints a list of locations stored in the graph.The function iterates through the locations array within the Graph structure.It prints the ID and name of each location in the format "ID. Name".

- **CheckCabInput():** The function checkCabInput is designed to validate and check the user input for the cab type. It ensures that the input matches one of the available cab types: mini, sedan, suv, bike, or auto. Additionally, it checks if the cab type corresponds to a specific cabChoice, and if not, it prompts the user again for a valid input.

- **displayest():**The function `displayest` calculates and returns the estimated fare for a specific cab type, given the distance traveled. It utilizes the fare rate of the cab type and applies an additional charge per kilometer beyond the initial 3 kilometers. The calculated fare is then returned as the estimated fare for the given distance and cab type.

- **readData():** The `readData` function opens a file called "drivers.txt" and reads driver information from it. It stores the data in an array of `Driver` structures. The function ensures that it reads the specified number of drivers and handles any file opening errors.

- **readLocations():** The `readLocations` function opens a file called "locations.txt" and reads location information from it. It stores the data in an array of `Location` structures. The function ensures that it reads the specified number of locations and handles any file opening errors.

- **haversine():** The function haversine calculates the great-circle distance between two points on the Earth's surface using their latitude and longitude coordinates. It implements the Haversine formula, which is an approximation formula used to determine distances between two points on a sphere.The inputs to the function are the latitude and longitude coordinates of two points (lat1, lon1, lat2, lon2). The function calculates the angular distances between the latitudes and longitudes, converts them to radians, and applies the Haversine formula to compute the central angle (c) between the two points. This central angle represents the angular distance along the surface of the Earth.The function then multiplies the central angle by the Earth's radius (assumed to be 6371 kilometers) to obtain the distance between the two points in kilometers.Finally, the calculated distance is returned as the result of the function.This function is commonly used in geographic calculations to estimate distances between two points on the Earth's surface based on their latitude and longitude coordinates.

- **initGraph():**The initGraph function sets up the Graph structure by reading location and driver data, assigning cab types to drivers, and initializing the distance matrix with initial values. This prepares the graph for further operations and calculations.

- **dijkstra():** The dijkstra function implements Dijkstra's algorithm to find the shortest path in a graph from a given source vertex (src) to a destination vertex (dest). It also provides information about the path, including the locations visited and the estimated time for the drop.

- **minDistance():** The minDistance function helps in finding the vertex(represents a location or a node) with the minimum distance value among the vertices that have not been included in the shortest path tree. It is commonly used in graph algorithms like Dijkstra's algorithm to determine the next vertex to be included in the shortest path calculation.

- i**nitializeEdges():-** The ‘initialiseEdges’ function plays a vital role in constructing the road network by creating connections between various locations. The function accepts a 2D matrix called matrix and an array of Location structures called locations. Using the addEdge function, the code defines multiple connections between different locations by specifying their indices and the locations array. Each connection represents a road or route between two locations in the transportation network. By invoking addEdge with different combinations of source and destination locations, the function establishes a comprehensive network of connections, allowing for efficient traversal and route calculations. The resulting matrix serves as a representation of the road network, enabling subsequent analyses such as pathfinding algorithms and distance calculations. Through the ‘initialiseEdges’ function, the road network is systematically built, facilitating the management and optimization of transportation within the system.

- **findClosestDriver():-** The ‘findClosestDriver’ function aims to locate the index of the nearest driver based on a specified location and cab type. Initially, the function declares variables for drivers and numDrivers to store driver information and their count, respectively. The code then verifies the specified cab type using string comparisons (strcmp) and assigns the corresponding driver array and the number of drivers from the graph structure. If an invalid cab type is provided, an error message is displayed, and -1 is returned. The function checks if there are available drivers for the specified cab type, and if not, an appropriate message is printed, and -1 is returned. Subsequently, the function initializes variables for tracking the minimum distance (minDistance) and the index of the closest driver (closestDriverIndex). By iterating through the drivers of the specified cab type, the function employs the Haversine formula to calculate the distance between the specified location and the driver's latitude and longitude. Whenever a shorter distance is found, the minimum distance and closest driver index are updated accordingly. After evaluating all drivers, the index of the closest driver is returned. In summary, the ‘findClosestDriver’ function utilizes the Haversine formula to identify the nearest driver based on the given location and cab type within the provided graph structure.

- **assignCabTypes():** The function assignCabTypes assigns the appropriate cab types to drivers based on the specified limits for each cab type. It iterates through the drivers' array and assigns them to the corresponding cab type arrays (Mini, Sedan, SUV, Bike, Auto) within the Graph structure. It also updates the counts of drivers for each cab type (numMini, numSedan, numSuv, numBike, numAuto).If the number of drivers exceeds the specified limits for all cab types, the loop breaks to avoid assigning drivers beyond the specified limits.At the end of the function, the counts of drivers for each cab type are updated in the Graph structure.

- **allotCab():-** The ‘allotCab’ function plays a crucial role in the process of assigning a cab to a customer based on the provided pickup and drop-off locations, as well as the specified cab type.

  →Initially, the function adjusts the pickup and drop-off location indices to match the zero-based indexing used in the graph data structure.

  →It then checks the validity of the cab type by calling the ‘isValidCabType’ function and exits if the cab type is invalid. Next, the function calls ‘findClosestDriver’ to determine the index of the closest available driver for the specified cab type and pickup location.

  →If no driver is found, an appropriate message is displayed, and the function terminates. Upon locating an available driver, a success message is printed, indicating that a cab of the specified type has been allotted.

  →The driver's details are accessed based on the cab type, and the function calculates the distance and estimated time of arrival (ETA) to the pickup location using the ‘haversine’ function.

  →The ETA is then assigned to the driver's etaToPickup field.

  →Finally, the function prints the driver's details, including the ID, name, phone number, car number, car type,driver’s rating and estimated time of cab arrival (ETA) to the pickup location.

  Overall, the ‘allotCab’ function manages the process of assigning a suitable driver and conveying essential details to the customer, facilitating a smooth cab allotment experience

ADVANCE BOOKING MODULE

The Advance Booking Module facilitates pre-scheduled cab assignments for future rides, offering customers the convenience of planning and securing their rides in advance. It consists of the following functions:

- **isBookingTimeValid():-**
  The isBookingTimeValid function checks the validity of a requested booking time. It starts by obtaining the current time in IST (Indian

Standard Time) and adjusting it for the time zone difference. The function then parses the requested time, calculates the time difference between the requested time and the current time, taking into account the possibility of the requested time being on the next day. Finally, it checks if the time difference falls within the valid range of 2 to 20 hours. If the requested time satisfies these conditions, indicating a valid booking time, the function returns 1. Otherwise, it returns 0 to indicate an invalid booking time. By comparing the requested time with the current time, accounting for time zone and day changes, the function provides a mechanism to determine if a booking time is valid for further processing or not.

- **assigndriver():-**
  The ‘assignDriver’ function is responsible for assigning a driver for a requested cab type. It begins by declaring necessary variables such as i, ‘availableDriverIndices’, and ‘numAvailableDrivers’. The current time is obtained using ‘time(NULL)’. Based on the requested cab type, the function determines the corresponding filename by using a series of ‘strcmp’ comparisons. The file associated with the cab type is then opened for reading, and if the file opening fails, an error message is displayed, and the function returns. Next, the function reads the number of drivers from the file and allocates memory to store the driver details. The details of each driver are read from the file and stored in the ‘fileDrivers’ array. After closing the file, the function loops through the ‘fileDrivers’ array to find the available drivers for the requested cab type. The indices of available drivers are stored in the ‘availableDriverIndices’ array. If there are available drivers, one driver is randomly assigned and its details are printed. If no drivers are available for the requested cab type, an appropriate message is displayed. Finally, the dynamically allocated memory for ‘availableDriverIndices’ and ‘fileDrivers’ is freed to avoid memory leaks.

- **displayest()**
- **CheckCabInput()**
- **displayAvailableCabTypes()**
- **printLocations()**

BILL GENERATION MODULE

- **getSurgeFee()**: The function getSurgeFee() calculates the surge fee multiplier based on the current time of the day. If the current hour is between 9 AM to 11 AM or 7 PM to 9 PM, it returns a surge fee of 1.2 (which represents a 20% increase). Otherwise, it returns a surge fee of 1.0, indicating no surge fee is applied outside of the specified time slots.

9. IMPLEMENTATION

9.1 DATA ORGANISATION AND LANGUAGE CONSTRUCTS

- Arrays: Arrays are used to organize and manage collections of similar data elements. In the code, arrays are utilized to store information such as different cab types and various locations. By using arrays, the code can efficiently store and access multiple instances of the same data type.

- Structures: Structures provide a way to define custom data types that can hold different types of data under a single entity. They are used to group related data fields together. In the code, structures are employed to represent user profiles, cab types, and booking requests. This helps in organizing and managing complex data structures, making it easier to access and manipulate specific information.

- Array of Structures: An array of structures allows for the storage of multiple instances of structured data. In the code, this construct is used to store and manage information such as different cab types and driver details. By using an array of structures, the code can efficiently store and retrieve data related to each instance, enabling easier data management and manipulation.

- Files: Files are used to store data persistently, meaning that the data is retained even when the program is not running. In the code, files are used to store important information like user profiles, FAQs, and driver details. By utilizing files, the code ensures that data is not lost between program executions and can be easily accessed and modified. Files provide a reliable means of data storage and retrieval, contributing to the overall functionality of the code.

USER INTERFACE DESIGN

The user interface design in the code primarily relies on a command-line interface (CLI) approach, where users interact with the program by entering choices and providing input through the console. The UI is text-based and menu-driven, displaying options and prompts for user input.

The design provides a sequential flow where users are presented with menus and prompted to enter choices or provide required information. The UI handles user inputs, validates them, and performs appropriate actions based on the selected options. Information and feedback are displayed through text output in the console.

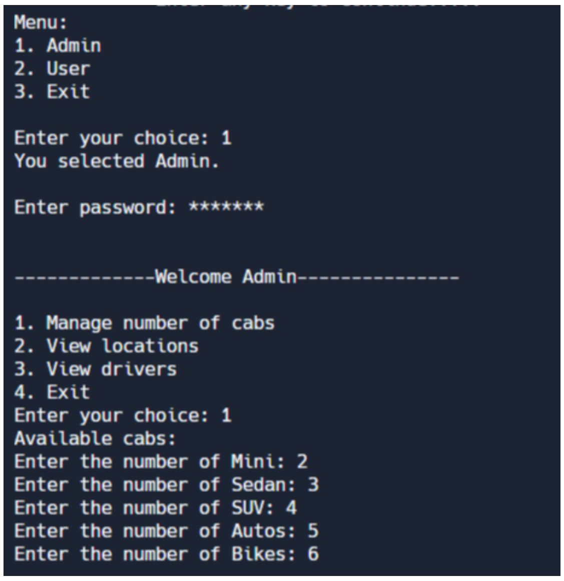
Overall, the user interface design aims to provide a simple and intuitive interaction model through text-based menus and prompts, allowing users to navigate the program and perform desired actions efficiently.

10. VALIDATION THROUGH DETAILED TEST CASES:

ADMIN MODULE

The first option provided is to go to the admin or the user interface. If the option chosen is admin, then the following happens:

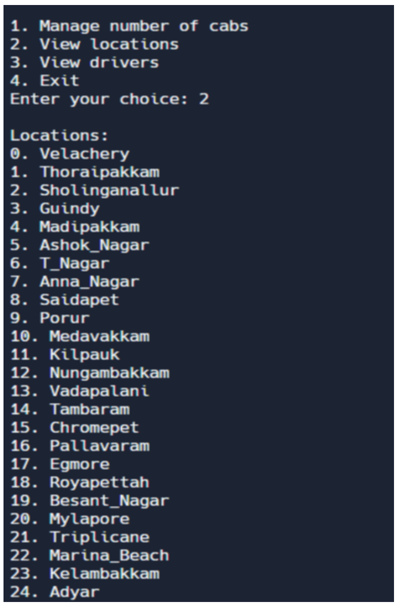
If the first option selected is to access the admin interface, a prompt will request the user to enter the password. If the entered password matches the correct password that has been set, the admin interface will be displayed.



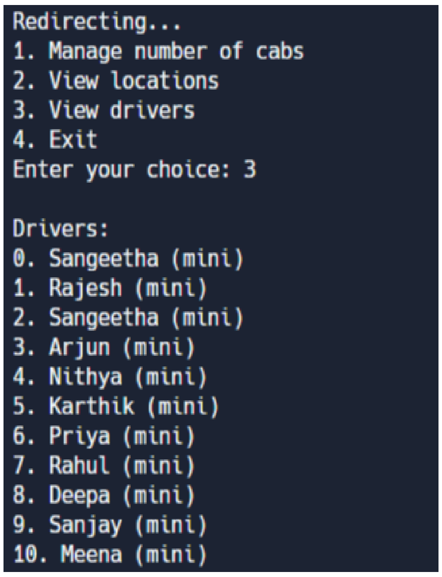
```
Menu:
1. Admin
2. User
3. Exit

Enter your choice: 1
You selected Admin.

Enter password: *******


------------Welcome Admin--------------

1. Manage number of cabs
2. View locations
3. View drivers
4. Exit
Enter your choice: 1
Available cabs:
Enter the number of Mini: 2
Enter the number of Sedan: 3
Enter the number of SUV: 4
Enter the number of Autos: 5
Enter the number of Bikes: 6
```

Once inside the admin interface, the user will be presented with a menu. If option 1, "Manage number of cabs," is chosen, the system will prompt the user to input the number of cabs for each type. After taking the input, the user will be redirected back to the admin interface.

Alternatively, if option 2, "View locations," or option 3, "View drivers," is selected, the system will retrieve the relevant data from the database and display the drivers' information or the stored locations to the admin in the following format.



```
1. Manage number of cabs
2. View locations
3. View drivers
4. Exit
Enter your choice: 2

Locations:
0. Velachery
1. Thoraipakkam
2. Sholinganallur
3. Guindy
4. Madipakkam
5. Ashok_Nagar
6. T_Nagar
7. Anna_Nagar
8. Saidapet
9. Porur
10. Medavakkam
11. Kilpauk
12. Nungambakkam
13. Vadapalani
14. Tambaram
15. Chromepet
16. Pallavaram
17. Egmore
18. Royapettah
19. Besant_Nagar
20. Mylapore
21. Triplicane
22. Marina_Beach
23. Kelambakkam
24. Adyar
```



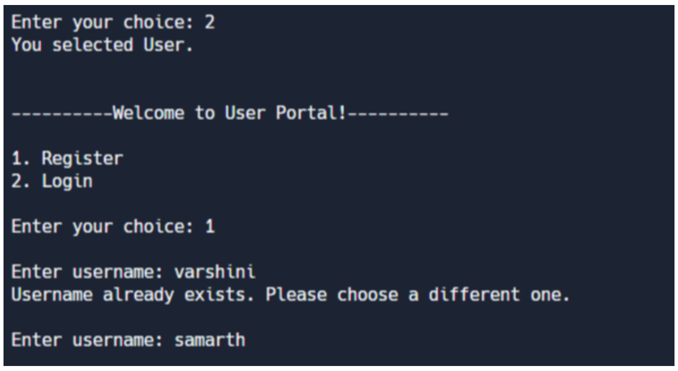
```
Redirecting...
1. Manage number of cabs
2. View locations
3. View drivers
4. Exit
Enter your choice: 3

Drivers:
0. Sangeetha (mini)
1. Rajesh (mini)
2. Sangeetha (mini)
3. Arjun (mini)
4. Nithya (mini)
5. Karthik (mini)
6. Priya (mini)
7. Rahul (mini)
8. Deepa (mini)
9. Sanjay (mini)
10. Meena (mini)
```

If the user chooses option 4 , “Exit”, then the user is taken back to the main menu.

USER MODULE

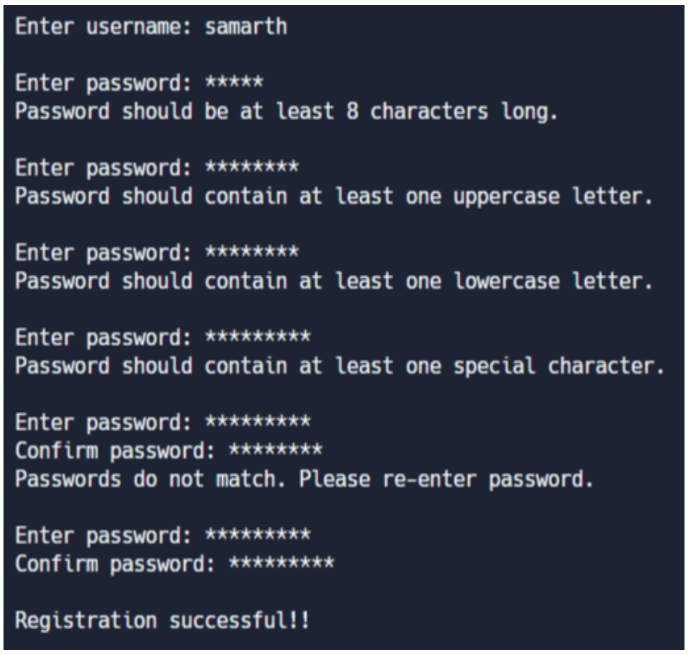
If the user selects the "User" option from the main menu, the system will provide them with two choices: to register or to log in. If the user chooses to register, the following steps will be taken:

1. The system will prompt the user to enter a username. To ensure uniqueness, the code will verify if the entered username already exists in the system. If it does, the user will be asked to re-enter a different username.



```
Enter your choice: 2
You selected User.


---------Welcome to User Portal!---------

1. Register
2. Login

Enter your choice: 1

Enter username: varshini
Username already exists. Please choose a different one.

Enter username: samarth
```

2. After verifying the uniqueness of the username, the user will be prompted to enter a password. The system will enforce the requirement for a strong password, which typically includes a combination of uppercase and lowercase letters, numbers, and special characters. The user will be asked to re-enter the password until the conditions for a strong password are met.

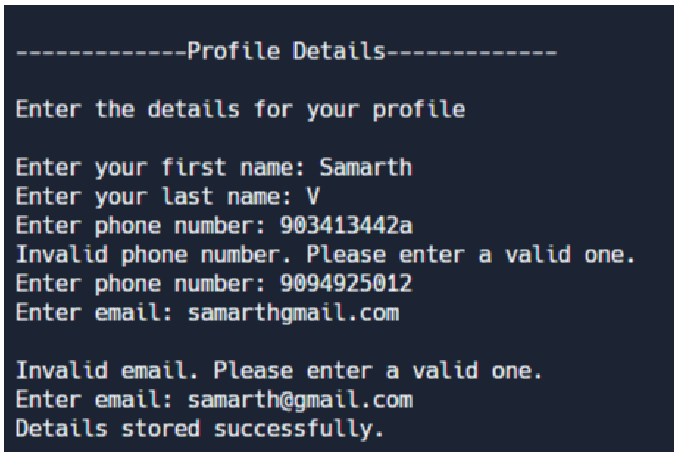


```
Enter username: samarth

Enter password: *****
Password should be at least 8 characters long.

Enter password: ********
Password should contain at least one uppercase letter.

Enter password: ********
Password should contain at least one lowercase letter.

Enter password: *********
Password should contain at least one special character.

Enter password: *********
Confirm password: ********
Passwords do not match. Please re-enter password.

Enter password: *********
Confirm password: *********

Registration successful!!
```

3. Next, the user will be asked to re-enter the password for confirmation. If the initially entered password and the confirmed password do not match, the system will prompt the user to re-enter the password and confirm it correctly.

4. Once all the conditions are satisfied, the system will confirm that the user has been successfully registered.

5. After successful registration, the user is prompted to enter the details for their profile, such as name, email and phone number and they are checked for their validity as well.



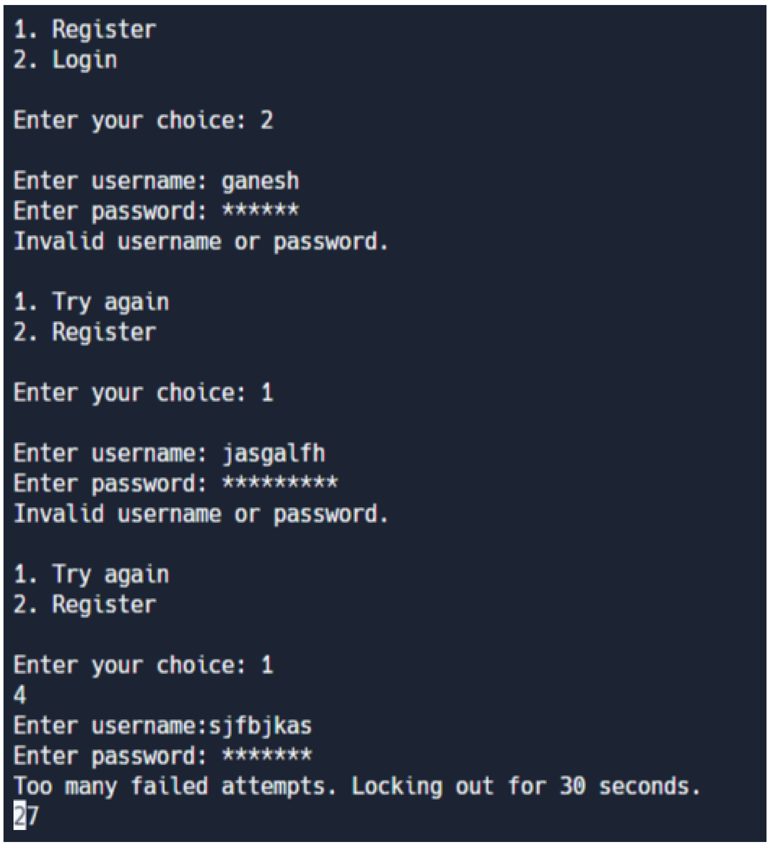
```
-------------Profile Details-------------

Enter the details for your profile

Enter your first name: Samarth
Enter your last name: V
Enter phone number: 903413442a
Invalid phone number. Please enter a valid one.
Enter phone number: 9094925012
Enter email: samarthgmail.com

Invalid email. Please enter a valid one.
Enter email: samarth@gmail.com
Details stored successfully.
```

6. The user is directed back to the user portal after the details are stored.

If the user chooses to log in, the following steps will be taken:

1. The user will be prompted to enter their username and password. The system will verify if the provided username and password combination exists in the system. If they don't match or do not exist, the user will be given the option to try again or proceed with registration.

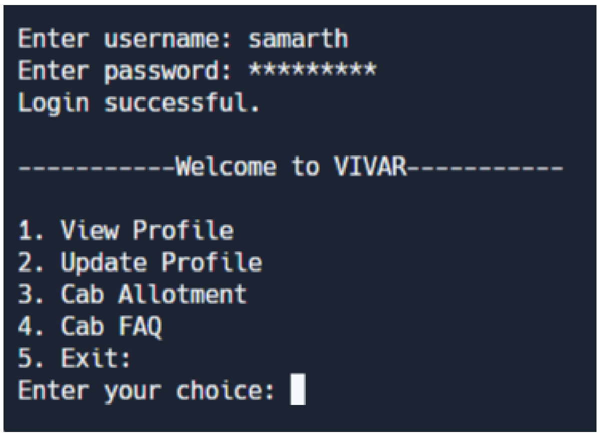
2. For security purposes, a maximum of three incorrect login attempts will be allowed. If the user exceeds this limit, a temporary lockout of 30 seconds will be initiated. After the lockout period ends, the user can attempt to log in again.



```
1. Register
2. Login

Enter your choice: 2

Enter username: ganesh
Enter password: ******
Invalid username or password.

1. Try again
2. Register

Enter your choice: 1

Enter username: jasgalfh
Enter password: *********
Invalid username or password.

1. Try again
2. Register

Enter your choice: 1
4
Enter username:sjfbjkas
Enter password: *******
Too many failed attempts. Locking out for 30 seconds.
27
```

3. If the user chooses to register instead of logging in, they will be directed to the registration page. Here, they can create a new profile by providing the required information and following the registration process.

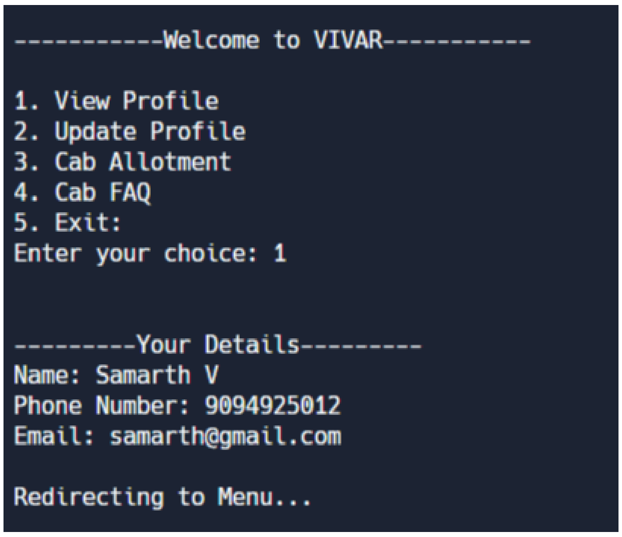
Upon successful login, the user will be directed to their profile page.



```
Enter username: samarth
Enter password: *********
Login successful.

-----------Welcome to VIVAR-----------

1. View Profile
2. Update Profile
3. Cab Allotment
4. Cab FAQ
5. Exit:
Enter your choice: 
```

PROFILE MODULE

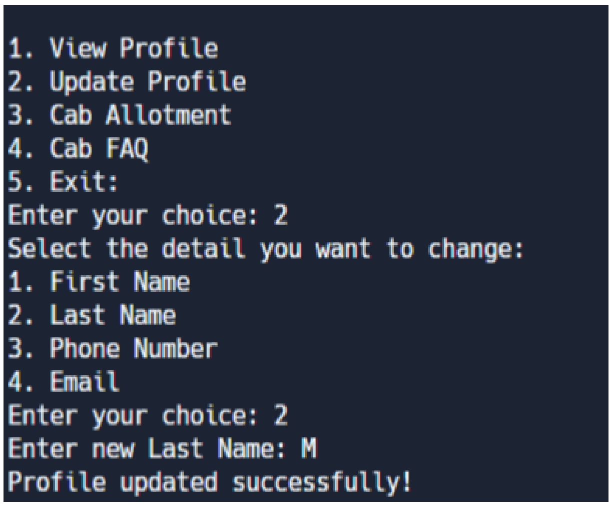
The user is led to the profile page, where they can choose to view or edit their profile, access general FAQs, and go to the cab allotment section. If the user chooses the view profile option, the following happens:



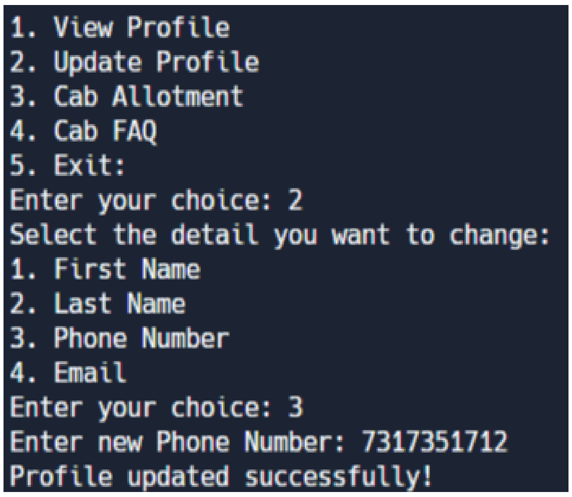
```
-----------Welcome to VIVAR-----------

1. View Profile
2. Update Profile
3. Cab Allotment
4. Cab FAQ
5. Exit:
Enter your choice: 1


---------Your Details---------
Name: Samarth V
Phone Number: 9094925012
Email: samarth@gmail.com

Redirecting to Menu...
```

The details of the user stored are displayed and the user is redirected back to the profile page.

If the user chooses the update profile option, the user is given an option to change their details that they entered at the time of registration.



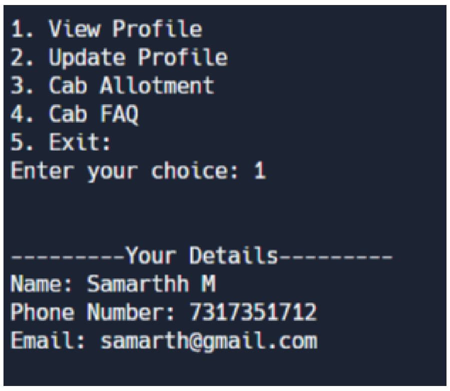
```
1. View Profile
2. Update Profile
3. Cab Allotment
4. Cab FAQ
5. Exit:
Enter your choice: 2
Select the detail you want to change:
1. First Name
2. Last Name
3. Phone Number
4. Email
Enter your choice: 2
Enter new Last Name: M
Profile updated successfully!
```



```
1. View Profile
2. Update Profile
3. Cab Allotment
4. Cab FAQ
5. Exit:
Enter your choice: 2
Select the detail you want to change:
1. First Name
2. Last Name
3. Phone Number
4. Email
Enter your choice: 3
Enter new Phone Number: 7317351712
Profile updated successfully!
```

In the above pictures, the last name and the email have been updated.

The updated details can be viewed by choosing the view profile option:



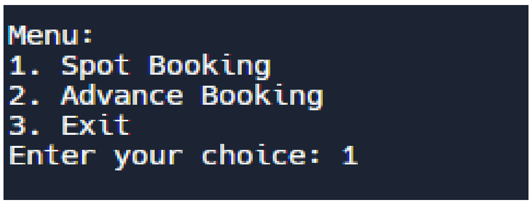
```
1. View Profile
2. Update Profile
3. Cab Allotment
4. Cab FAQ
5. Exit:
Enter your choice: 1


---------Your Details---------
Name: Samarthh M
Phone Number: 7317351712
Email: samarth@gmail.com
```

If the user chooses the Cab FAQ option, the user can view the general FAQs related to cab allotment and the interface.

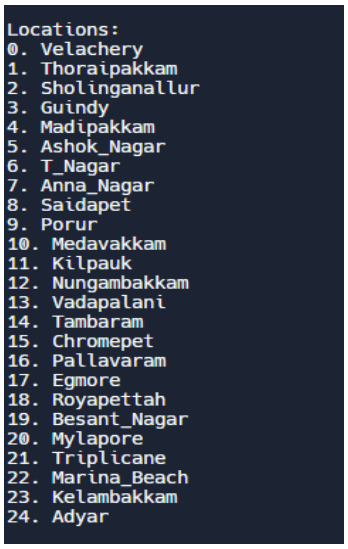
CAB ALLOTMENT

SPOT BOOKING MODULE:

1. Once a user logs into their account , the user is asked about the type of booking they would prefer. If the user chooses spot booking i.e 1 from the menu , 25 locations are displayed providing the user the choice for pick-up and drop- location.



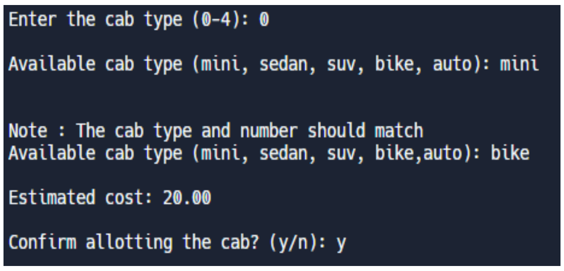
```
Menu:
1. Spot Booking
2. Advance Booking
3. Exit
Enter your choice: 1
```



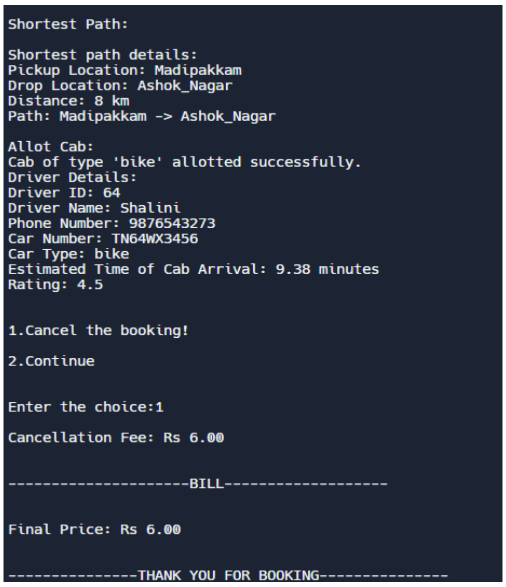
```
Locations:
0. Velachery
1. Thoraipakkam
2. Sholinganallur
3. Guindy
4. Madipakkam
5. Ashok_Nagar
6. T_Nagar
7. Anna_Nagar
8. Saidapet
9. Porur
10. Medavakkam
11. Kilpauk
12. Nungambakkam
13. Vadapalani
14. Tambaram
15. Chromepet
16. Pallavaram
17. Egmore
18. Royapettah
19. Besant_Nagar
20. Mylapore
21. Triplicane
22. Marina_Beach
23. Kelambakkam
24. Adyar
```

Available types of cab are displayed along with the trip fare. Now , the user is asked to select the cab by entering 0 for bike,1 for auto,2 for suv,3 for mini and 4 for sedan.

User is again asked for the cab type, and ensures that the cabtype entered and previously selected cab is matching. In case they don't match a message is displayed and the user is asked to enter the type again.The estimated cost of the trip is displayed, once the cab is selected.



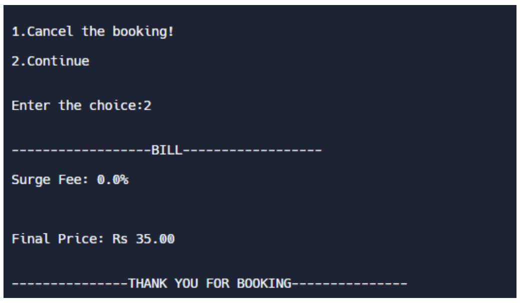
```
Enter the cab type (0-4): 0

Available cab type (mini, sedan, suv, bike, auto): mini


Note : The cab type and number should match
Available cab type (mini, sedan, suv, bike,auto): bike

Estimated cost: 20.00

Confirm allotting the cab? (y/n): y
```



```
Shortest Path:

Shortest path details:
Pickup Location: Madipakkam
Drop Location: Ashok_Nagar
Distance: 8 km
Path: Madipakkam -> Ashok_Nagar

Allot Cab:
Cab of type 'bike' allotted successfully.
Driver Details:
Driver ID: 64
Driver Name: Shalini
Phone Number: 9876543273
Car Number: TN64WX3456
Car Type: bike
Estimated Time of Cab Arrival: 9.38 minutes
Rating: 4.5


1.Cancel the booking!

2.Continue


Enter the choice:1

Cancellation Fee: Rs 6.00


--------------------BILL------------------


Final Price: Rs 6.00


--------------THANK YOU FOR BOOKING--------------
```

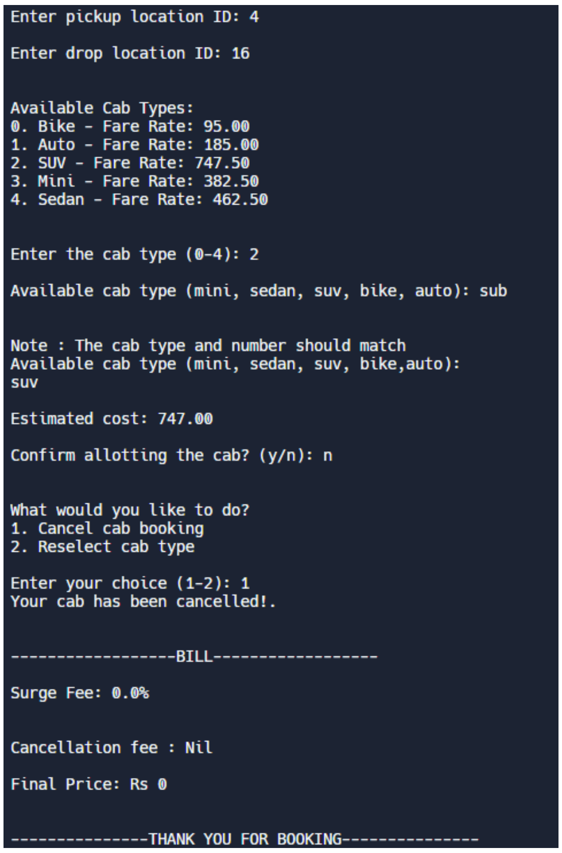
The user is asked for confirmation of the ride. If opted yes, we display the shortest path for the trip along with driver details (for eg name, driver id, phone number, number plate) . The estimated time of cab arrival is also displayed.

Now two options - (i) cancel cab or (ii) continue with booking , are displayed. If user enters 1, the cancellation fee is printed along with the final bill



```
1.Cancel the booking!
2.Continue

Enter the choice:2

-----------------BILL-----------------
Surge Fee: 0.0%


Final Price: Rs 35.00

--------------THANK YOU FOR BOOKING--------------
```

If the user opts for option 2 (i.e continue) the final fare including the surge fee is printed. The surge fee is calculated when cab is booked during the surge hours (i.e 9-11 am , 7-9pm)



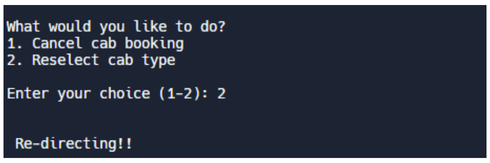
```
Enter pickup location ID: 4

Enter drop location ID: 16


Available Cab Types:
0. Bike - Fare Rate: 95.00
1. Auto - Fare Rate: 185.00
2. SUV - Fare Rate: 747.50
3. Mini - Fare Rate: 382.50
4. Sedan - Fare Rate: 462.50


Enter the cab type (0-4): 2

Available cab type (mini, sedan, suv, bike, auto): sub


Note : The cab type and number should match
Available cab type (mini, sedan, suv, bike,auto):
suv

Estimated cost: 747.00

Confirm allotting the cab? (y/n): n


What would you like to do?
1. Cancel cab booking
2. Reselect cab type

Enter your choice (1-2): 1
Your cab has been cancelled!.


------------------BILL------------------

Surge Fee: 0.0%


Cancellation fee : Nil

Final Price: Rs 0


--------------THANK YOU FOR BOOKING--------------
```

In case the user denies confirmation of the cab, user is provided with the option to reselect the cab or to cancel the booking.

A message stating what the user would like to do is displayed.

If the user selects option 1(cancel the cab booking): A message saying booking has been cancelled is printed . The final bill in this case , will not include any cancellation or surge fee, hence final fare would be equal to Rs 0.

But incase user selects option 2(Reselection of cab) , the user is redirected

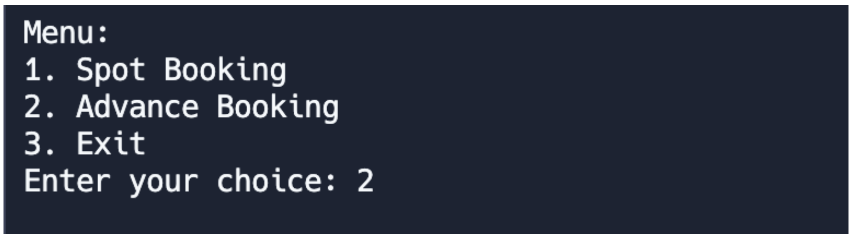


```
What would you like to do?
1. Cancel cab booking
2. Reselect cab type

Enter your choice (1-2): 2


 Re-directing!!
```

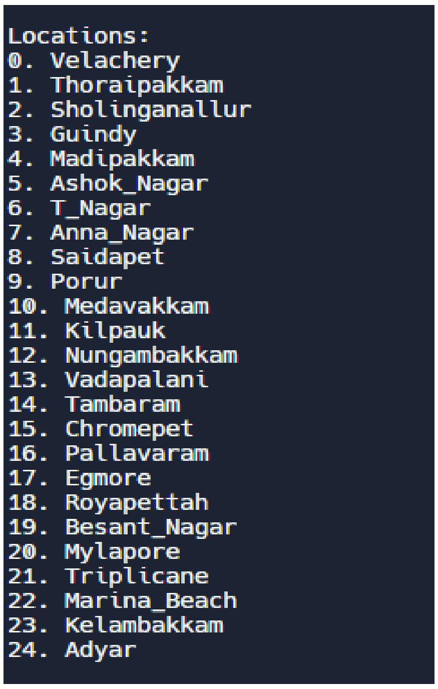
Again the different cab types along with the fare for that trip is displayed. Users are allowed to select the cab of their choice. The estimated cost is printed and the shortest path along with driver details are displayed. The final fare is generated at the end .

ADVANCE BOOKING MODULE:

Once a user logs into their account , the user is asked about the type of booking they would prefer. If the user chooses advance booking i.e 2 from the menu , 25 locations are displayed providing the user the choice for pick-up and drop- location.

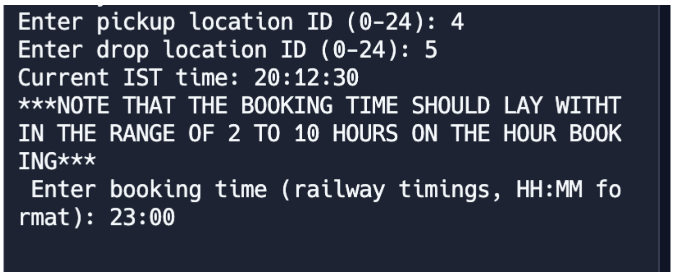


```
Menu:
1. Spot Booking
2. Advance Booking
3. Exit
Enter your choice: 2
```



```
Locations:
0. Velachery
1. Thoraipakkam
2. Sholinganallur
3. Guindy
4. Madipakkam
5. Ashok_Nagar
6. T_Nagar
7. Anna_Nagar
8. Saidapet
9. Porur
10. Medavakkam
11. Kilpauk
12. Nungambakkam
13. Vadapalani
14. Tambaram
15. Chromepet
16. Pallavaram
17. Egmore
18. Royapettah
19. Besant_Nagar
20. Mylapore
21. Triplicane
22. Marina_Beach
23. Kelambakkam
24. Adyar
```

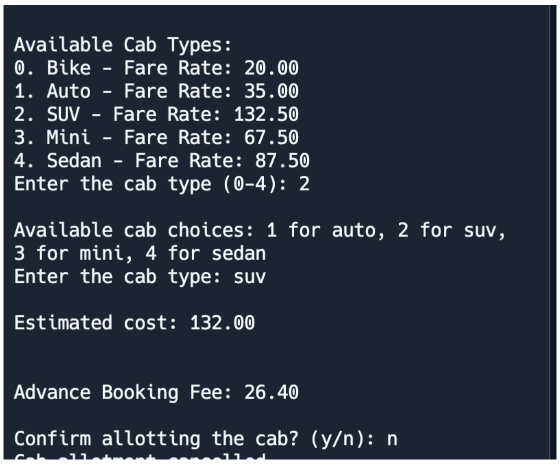
Then the current IST time is displayed for reference. Then the user is asked to enter the booking time. If the booking time does not lay within 2 to 20 hours of the current time. It will display as invalid timing. So,once the user enters the valid booking time.



```
Enter pickup location ID (0-24): 4
Enter drop location ID (0-24): 5
Current IST time: 20:12:30
***NOTE THAT THE BOOKING TIME SHOULD LAY WITHT
IN THE RANGE OF 2 TO 10 HOURS ON THE HOUR BOOK
ING***
 Enter booking time (railway timings, HH:MM fo
rmat): 23:00
```

Available types of cab are displayed along with the trip fare. Now , the user is asked to select the cab by entering 0 for bike,1 for auto,2 for suv,3 for mini and 4 for sedan.

User is again asked for the cab type, and ensures that the cabtype entered and previously selected cab is matching.



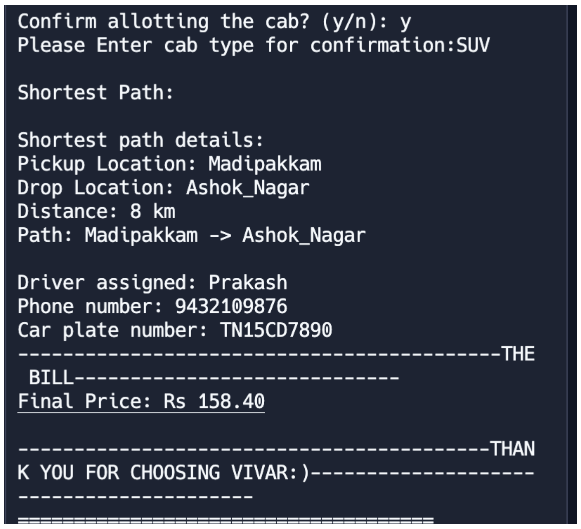
```
Available Cab Types:
0. Bike - Fare Rate: 20.00
1. Auto - Fare Rate: 35.00
2. SUV - Fare Rate: 132.50
3. Mini - Fare Rate: 67.50
4. Sedan - Fare Rate: 87.50
Enter the cab type (0-4): 2

Available cab choices: 1 for auto, 2 for suv,
3 for mini, 4 for sedan
Enter the cab type: suv

Estimated cost: 132.00


Advance Booking Fee: 26.40

Confirm allotting the cab? (y/n): n
```

Incase they don't match a message is displayed and user is asked to enter the type again

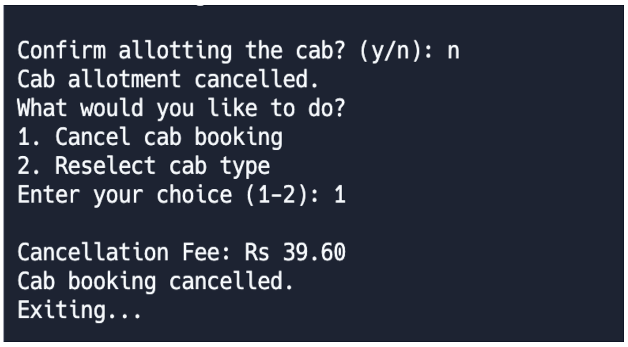
The estimated cost and advance booking fee of the trip is displayed, once the cab is selected .

The user is asked for confirmation of the ride. If opted yes, then cabType will be asked for confirmation once the cabType is entered.We display the shortest path for trip along with driver details (for eg name, driver id, phone number, number plate). The estimated time of cab arrival is also displayed. Along with the final bill price

Now two options - 1 cancel cab or 2 reselect cab type , are displayed. If user enters 1, the cancellation fee is printed along with the final bill



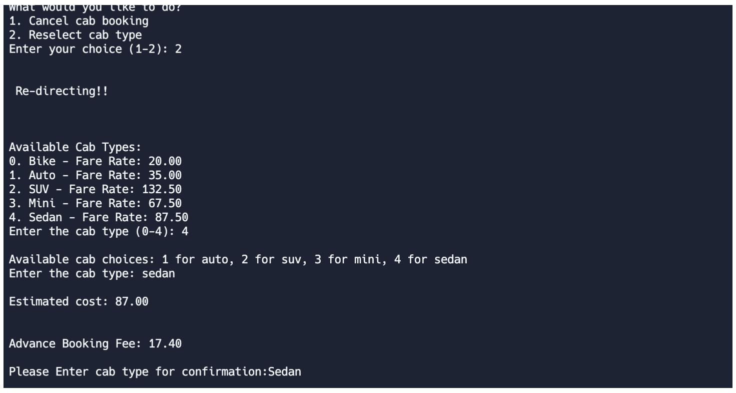
```
Confirm allotting the cab? (y/n): y
Please Enter cab type for confirmation:SUV

Shortest Path:

Shortest path details:
Pickup Location: Madipakkam
Drop Location: Ashok_Nagar
Distance: 8 km
Path: Madipakkam -> Ashok_Nagar

Driver assigned: Prakash
Phone number: 9432109876
Car plate number: TN15CD7890
------------------------------------------THE
 BILL-----------------------------
Final Price: Rs 158.40

------------------------------------------THAN
K YOU FOR CHOOSING VIVAR:)--------------------
---------------------
```



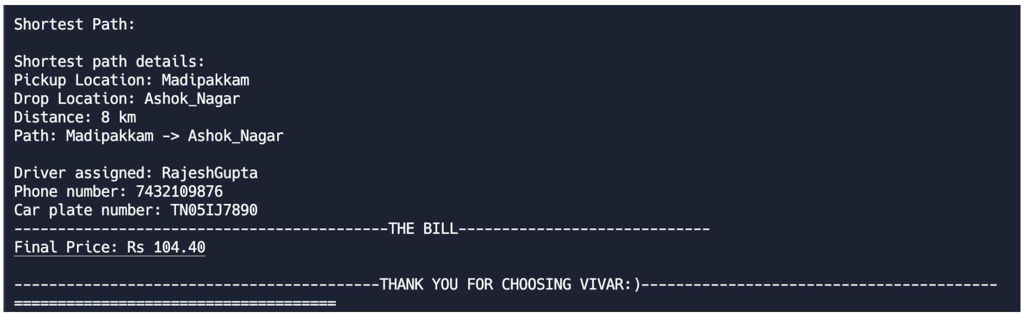
```
Confirm allotting the cab? (y/n): n
Cab allotment cancelled.
What would you like to do?
1. Cancel cab booking
2. Reselect cab type
Enter your choice (1-2): 1

Cancellation Fee: Rs 39.60
Cab booking cancelled.
Exiting...
```

But incase user selects option 2(Reselection of cab) , the user is redirected

Again the different cab types along with the fare for that trip is displayed. Users are allowed to select the cab of their choice. The estimated cost is printed and the shortest path along with driver details are displayed. The final fare is generated at the end .



```
What would you like to do?
1. Cancel cab booking
2. Reselect cab type
Enter your choice (1-2): 2


 Re-directing!!



Available Cab Types:
0. Bike - Fare Rate: 20.00
1. Auto - Fare Rate: 35.00
2. SUV - Fare Rate: 132.50
3. Mini - Fare Rate: 67.50
4. Sedan - Fare Rate: 87.50
Enter the cab type (0-4): 4

Available cab choices: 1 for auto, 2 for suv, 3 for mini, 4 for sedan
Enter the cab type: sedan

Estimated cost: 87.00


Advance Booking Fee: 17.40

Please Enter cab type for confirmation:Sedan
```



```
Shortest Path:

Shortest path details:
Pickup Location: Madipakkam
Drop Location: Ashok_Nagar
Distance: 8 km
Path: Madipakkam -> Ashok_Nagar

Driver assigned: RajeshGupta
Phone number: 7432109876
Car plate number: TN05IJ7890
-------------------------------------------THE BILL-----------------------------
Final Price: Rs 104.40

------------------------------------------THANK YOU FOR CHOOSING VIVAR:)-----------------------------------------
=====================================
```

11. LIMITATIONS OF SOLUTION PROVIDED

USER AND PROFILE MODULE

1. **Lack of Input Validation**: The code does not perform extensive input validation for user inputs, such as sanitizing inputs to prevent security vulnerabilities like SQL injection or cross-site scripting (XSS) attacks. A robust software system would implement comprehensive input validation to ensure data integrity and security.

2. **Security Considerations**: While the code includes masking the password input during registration and login, it does not employ additional security measures like hashing and salting passwords for secure storage.

3. **Data Persistence** : The code uses a simple file-based approach for storing user data instead of more sophisticated and reliable persistence mechanisms, such as databases or cloud storage solutions.

4. **Scalability**: efficient data structures, algorithms, and database optimizations would be implemented to handle large user bases and high traffic loads effectively.

SPOT BOOKING MODULE

1. **Fixed Locations**: The code assumes a fixed set of 25 locations (0-24) in the graph. The user can choose pickup and drop locations only within this fixed set.

2. **Driver allotment:** The Driver for advance booking is allotted randomly based on the cab type. This is done,because we didn't consider the driver's side of the software. So, the driver can't accept the customer's advance booking request. And hence, the bookings are done randomly

3. **Availability of Drivers**: The code assumes that the availability of drivers for each cab type is fixed and not dynamically updated. The availability is determined by the numDrivers variables in the Graph structure.

ADVANCE BOOKING MODULE

1. **Driver allotment:-** The Driver for advance booking is allotted randomly based on the cab type. This is done,because we didn't consider the driver's side of the software. So, the driver can't accept the customer's advance booking request. And hence, the bookings are done randomly.

2. **Fixed Locations**: The code assumes a fixed set of 25 locations (0-24) in the graph. The user can choose pickup and drop locations only within this fixed set.

12. OBSERVATIONS WITH RESPECT TO SOCIETY, LEGAL AND ETHICAL AND ENVIRONMENTAL PERSPECTIVES

SOCIETY

- **Impact on local transportation industry:** Cab hailing websites may disrupt traditional taxi and transportation industries, potentially leading to job losses and other economic effects. Society may have concerns about the impact of cab hailing services on local transportation industries and their workers.

- **Availability of transportation options:** Cab hailing services can increase the availability of transportation options, particularly in areas where public transportation is limited. Society may view cab hailing websites as a positive force for providing greater access to transportation services.

- **Impact on traffic congestion:** Cab hailing services may contribute to increased traffic congestion, particularly in urban areas. Society may be concerned about the environmental impact of increased traffic and congestion, and the potential safety risks associated with more vehicles on the road.

LEGAL

- **Compliance with local regulations:** The website should ensure that it is compliant with local regulations and laws related to transportation and data privacy.

- **Liability and insurance**: The website should have proper liability insurance and legal agreements in place to protect the website and its users.

- **Intellectual property:** The website should respect intellectual property rights, such as trademarks and copyrights, and should not infringe on the intellectual property rights of others.

- **Contractual agreements:** The website should have clear contractual agreements in place with drivers, including provisions related to liability, insurance, and dispute resolution.

ETHICAL

- **Fair treatment of drivers:** The website should ensure that its drivers are treated fairly and are compensated fairly for their services.

- **Non-discrimination:** The website should not discriminate on the basis of race, gender, religion, sexual orientation, or other protected categories.

- **Environmental impact:** The website should consider the environmental impact of its services, and should take steps to reduce its carbon footprint, such as promoting the use of electric or hybrid vehicles.

- **Privacy and data ethics:** The website should respect user privacy and should not use user data in unethical ways, such as selling user data to third parties without user consent.

ENVIRONMENTAL

- **Carbon emissions**: The use of traditional fuel-powered cabs can contribute to carbon emissions and air pollution. Cab booking software should consider promoting or integrating eco-friendly options such as electric or hybrid vehicles, which have lower emissions and help reduce the environmental impact.

- **Efficient routing**: Optimized routing algorithms can minimize the distance traveled by cabs, reducing fuel consumption and emissions. The software system should prioritize assigning cabs based on proximity to the customer's location and consider traffic conditions to ensure efficient routes are taken.

- **Carpooling and ride-sharing**: Encouraging carpooling or ride-sharing options through the software system can significantly reduce the number of vehicles on the road. By matching customers with similar routes or nearby destinations, cab booking software can help reduce congestion and emissions associated with individual car trips.

- **Integration with public transportation**: By integrating cab booking software with public transportation systems, users can be provided with more sustainable travel options. This could include suggesting public transit routes for shorter distances or providing seamless transfers between public transportation and cab services

13. LEARNING OUTCOMES:

- **Effective Collaboration:** Team members learned how to collaborate effectively with each other to achieve a common goal. They developed communication skills, actively listened to each other's ideas, and worked cohesively to overcome challenges.

- **Project Management:** Through the team project, members gained experience in project management. They learned how to plan, organize, and prioritize tasks, ensuring that the project progressed smoothly and met its deadlines.

- **Technical Skills Improvement:** Working on the project exposed us to various technical challenges, which led to the improvement of their technical skills. They became more proficient in programming, problem-solving, and implementing algorithms.

- **Conflict Resolution:** In any collaborative effort, conflicts may arise. The team members learned how to address conflicts constructively, focusing on finding solutions and maintaining a positive team environment.

- **Version Control and Collaboration Tools:** We used version control systems (e.g., Git) and collaboration tools (e.g., Replit) to manage the project. This experience improved their understanding and proficiency in using these tools for future projects.

- **Presentation and Communication Skills:** We have enhanced our presentation and communication skills.We also learned how to articulate complex technical concepts to the review panel.

- **Time Management:** Successfully completing a team project required effective time management.We learned to set realistic deadlines, allocate time efficiently, and ensure progress throughout the project duration.

- **Team Bonding and Trust**: Working collaboratively on a team project fosters team bonding and trust. We developed a sense of camaraderie and learned to rely on each other's strengths.