Computer Science & Engineering Department

NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

April , 2025

# CAB BOOKING AND RENTAL MANAGEMENT SYSTEM

WORK REPORT

OOPs Project

**Team Members:**

**Bornil Gogoi (2312176)**

**Kunal Rajesh Sangalge (2312182)**

## Table of Contents

# PROJECT REPORT : Cab Booking System in C++

## ABSTRACT :

***"The future of transportation lies not in roads, but in smart systems that connect people with possibilities." — Elon Musk***

Transportation is not merely about moving people; it is about connecting lives, creating opportunities, and shaping progress. Yet in towns like Silchar — despite being home to a premier institute like NIT Silchar — access to organized, efficient transport remains limited, relying heavily on informal and fragmented systems. This creates barriers to mobility that ripple through daily life, affecting education, healthcare, commerce, and community well-being.

This project envisions a transformative step forward: a Cab Booking and Rental Management System, crafted in C++, designed specifically for the unique challenges and potentials of Silchar. Operating independently of internet connectivity, this lightweight, console-based platform empowers users to register securely, log in, and seamlessly book nearby cabs (Tuk-tuks and Taxis) or rent vehicles (Motorcycles, Hatchbacks, Sedans, SUVs) for any journey, short or long.

Through intelligent offline storage using CSV files and a dynamic, role-based authentication system for both users and drivers, the application reimagines what accessible transportation can look like in underserved regions. It brings mobility to the fingertips of students, and service providers alike — blending security, personalization, and real-world simulation through features such as dynamic driver location management, detailed booking histories, and flexible ride options including Solo and Shared modes.

Built on the pillars of Object-Oriented Programming (OOP) and powered by modern C++ techniques, the project demonstrates:

- Object lifecycle mastery through Constructors and Destructors,
- Hierarchical modelling via Inheritance and Polymorphism,
- Reusable, type-safe logic with Function Templates,
- Encapsulation ensuring data security and modularity,
- Robust Exception Handling safeguarding stability at every stage.

Precise geospatial calculations, enabled through the Haversine formula, allow the system to bridge the gap between simulation and reality, ensuring users connect with the most appropriate drivers based on true geographic proximity.

This project is more than a software application; it is a prototype for how inclusive, accessible technologies can reshape daily life, one journey at a time. It stands as a proof-of-concept that smart, context-aware systems — even built for offline settings — have the power to democratize mobility, empower communities, and lay the groundwork for a future where no one is left waiting at the roadside.

## PROBLEM STATEMENT :

Despite being a prominent academic and research hub in Assam, **Silchar** lacks the convenience and infrastructure of organized transport solutions that are widely available in larger metropolitan cities. Students and residents in and around **NIT Silchar** often face significant challenges when it comes to accessing safe, affordable, and reliable cab services or rental vehicles. The absence of structured booking systems means users must rely on informal channels, which may not always be dependable, efficient, or cost-transparent.

With an increasing student population, growing local commute demands, and the need for timely transportation to areas such as the railway station, market, or healthcare facilities, the necessity of an accessible cab booking platform becomes evident. However, due to constraints such as limited internet access, affordability, and the lack of dedicated service providers, existing app-based solutions (e.g., Ola, Uber) are not feasible or operational in the Silchar region.

This project addresses the following specific problem:

**How can we design and implement a reliable, offline, and user-friendly cab booking and vehicle rental system tailored to the transportation needs of Silchar and NIT Silchar residents. The system enables both users and drivers to register, manage profiles, update live locations, and track booking histories securely through a role-based platform.**

The system is:

- Simple enough to run on basic systems without requiring network access.
- Capable of handling secure user and driver registration, cab bookings, and vehicle rentals.
- Designed to store, retrieve, and update essential data securely using local file storage.
- Robust in handling incorrect inputs and guiding users with clear prompts.
- Scalable and adaptable for future integration with real-time services and broader enhancements.

In addition, the project aims to solve this problem while showcasing the practical use of advanced **C++ programming concepts**, including object-oriented design, error handling, and template programming.

## OBJECTIVE(S) :

The development of the Cab Booking and Rental Management System was guided by a vision to build a secure, reliable, and inclusive platform that empowers both users and drivers in Silchar to access organized transportation with confidence and ease.

The key objectives pursued in the project include:

1. **Design a Resilient, Console-Based User Interface:**
    - Craft an intuitive, menu-driven command-line interface (CLI) capable of operating seamlessly even in offline environments.
    - Structure the application flow to cater effortlessly to both first-time registrations and returning logins, ensuring a user-friendly experience for a wide range of users.
    - Support distinct registration and access paths for users and drivers, laying the groundwork for a role-specific service ecosystem.

2. **Establish Secure and Structured Identity Management:**
    - Implement a robust login and registration system fortified with password security mechanisms.
    - Enforce strict password complexity standards, mandating a minimum of 8 characters including uppercase letters, lowercase letters, and digits to safeguard user and driver accounts.
    - Protect sensitive credentials through masked input handling during password entry.
    - Enable comprehensive profile management functionalities, including profile editing and secure password changing, while maintaining strict input validation standards.
    - Ensure persistent storage of user and driver profiles in **Users.csv**, preserving data integrity and consistency.

3. **Develop Core Booking and Rental Operations:**
   - **Cab Booking**:
     - Display a curated list of locations with easy-to-navigate IDs.
     - Facilitate selection of pickup and drop-off points with dynamic distance computation using the Haversine formula.
     - Match users with nearby available drivers (Tuk-tuks and Taxis) within a 1 km radius to ensure rapid service.
     - Offer flexible booking options, empowering users to select between Solo and Shared rides, enabling cost efficiency.
     - Update driver geolocation dynamically after every booking to simulate real-world mobility patterns.
   - **Vehicle Rental**:
     - Provide users with a diverse range of rental vehicle options, including Motorcycles, Hatchbacks, Sedans, and SUVs.
     - Display vehicle specifications, owner details, and transparent pricing.
     - Accurately calculate total rental costs based on user-defined duration, ensuring financial clarity.

4. **Enable Data Persistence and Operational Continuity:**
   - Persistently store user, driver, location, vehicle, and booking data through efficient file management (CSV format).
   - Introduce daily driver location resets at 4 AM to realign operational accuracy, reflecting natural urban transport flows.

5. **Deliver Comprehensive History and Reporting Features:**
   - Maintain detailed and secure records of all booking and rental transactions, mapped individually to user profiles.
   - Provide tailored history views for both users and drivers, enhancing transparency and engagement.
   - Present historical data in professionally formatted, tabular layouts for clarity and ease of analysis.

6. **Apply and Showcase Robust Object-Oriented Programming (OOP) Practices:**
    - Apply **Encapsulation** to bundle data attributes and operations into secure, maintainable structures.
    - Employ **Inheritance** to model real-world hierarchies across different vehicle and booking types.
    - Utilize **Function Templates** for flexible and reusable input handling across varied data types.
    - Implement **Custom Exception Handling** to gracefully manage erroneous inputs, system failures, and authentication errors, promoting system resilience.

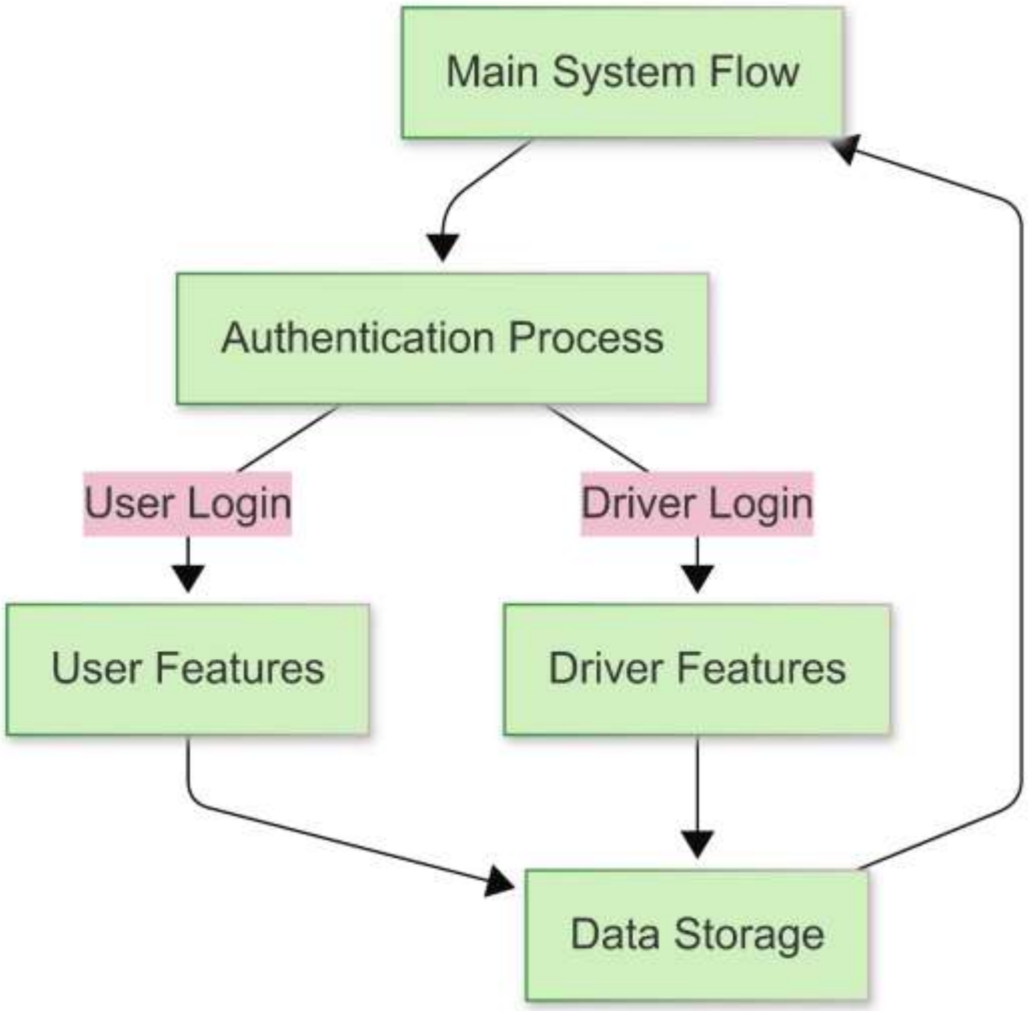
7. **Champion Input Validation and Error Handling for Security and Reliability:**
    - Validate critical inputs such as phone numbers, Aadhaar numbers, and email addresses rigorously to prevent data inconsistencies.
    - Handle authentication failures, login mismatches, and file handling errors through structured exception-based recovery, preserving system stability and user trust.

By pursuing these objectives, the project not only ensures a functional cab booking and rental service but also establishes a blueprint for building **secure, trustworthy, and locally optimized transport platforms** capable of scaling to meet broader regional needs.

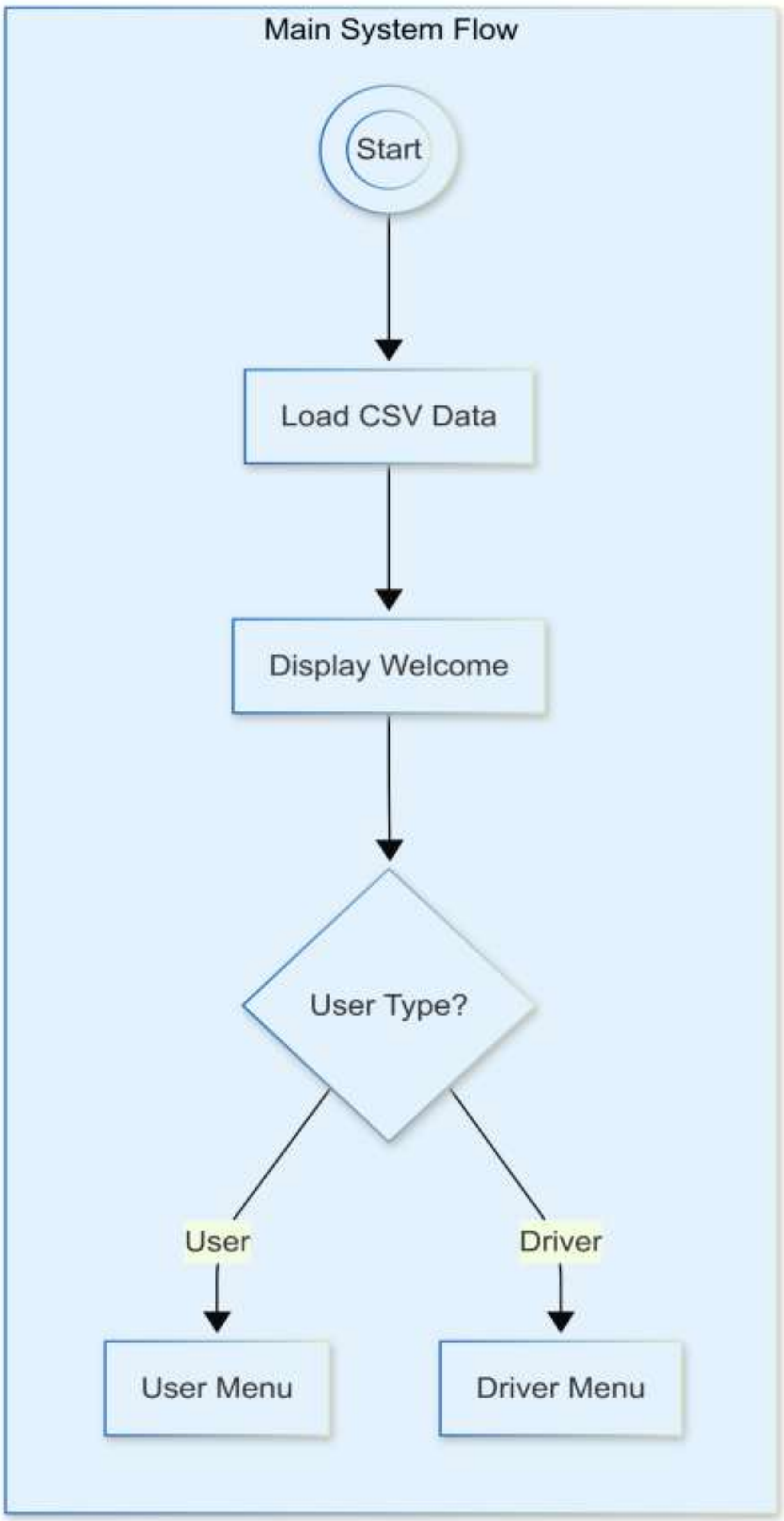
## WORK FLOW DIAGRAM :

The current flowchart is structured into distinct sections, each representing key functionalities within the system. The main flow starts with the authentication process, which branches into user and driver features. The user flow covers essential tasks such as booking a cab, renting a vehicle, viewing history, and editing personal information. Similarly, the driver flow focuses on tasks like viewing bookings, updating location, and editing personal details. Each section outlines the specific processes and interactions for users and drivers, ensuring a streamlined workflow. Additionally, the data flow ties everything together, ensuring that user and driver actions are appropriately saved and processed. This structure provides a comprehensive view of the system, emphasizing the interconnections between each feature while maintaining a clear, logical progression of operations.

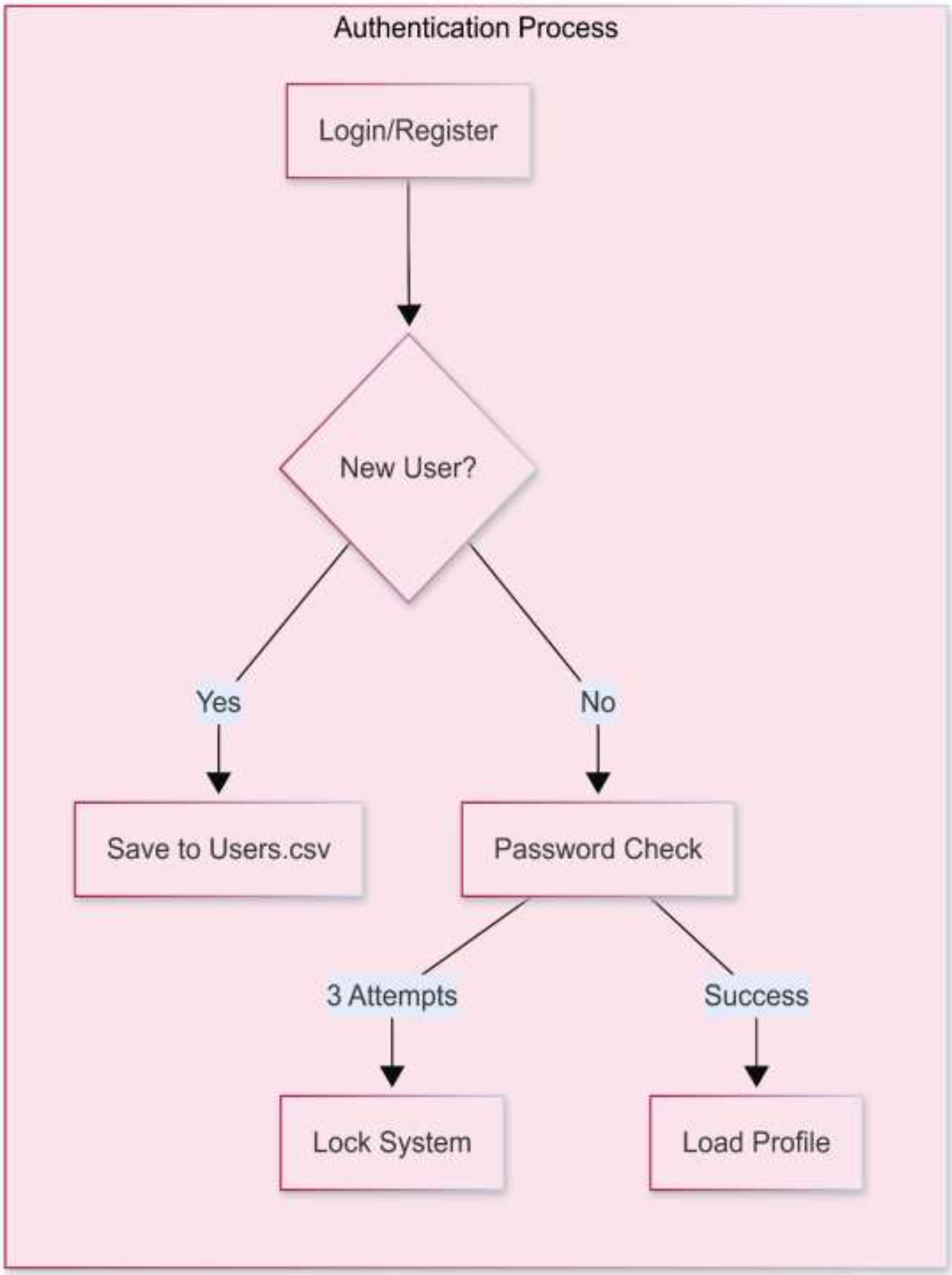
1) MASTER GRAPH :



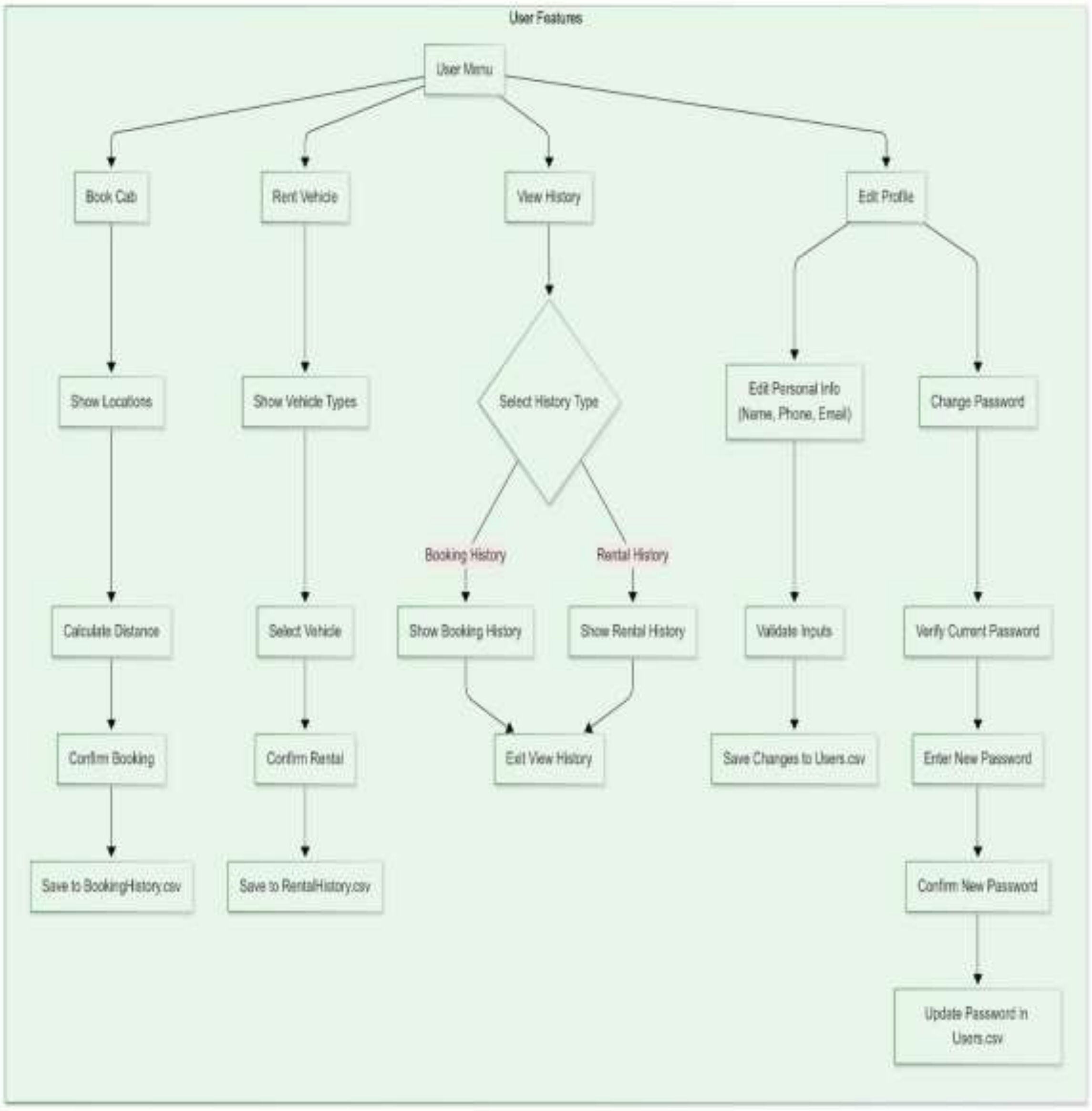
2) MAIN SYSTEM FLOW :



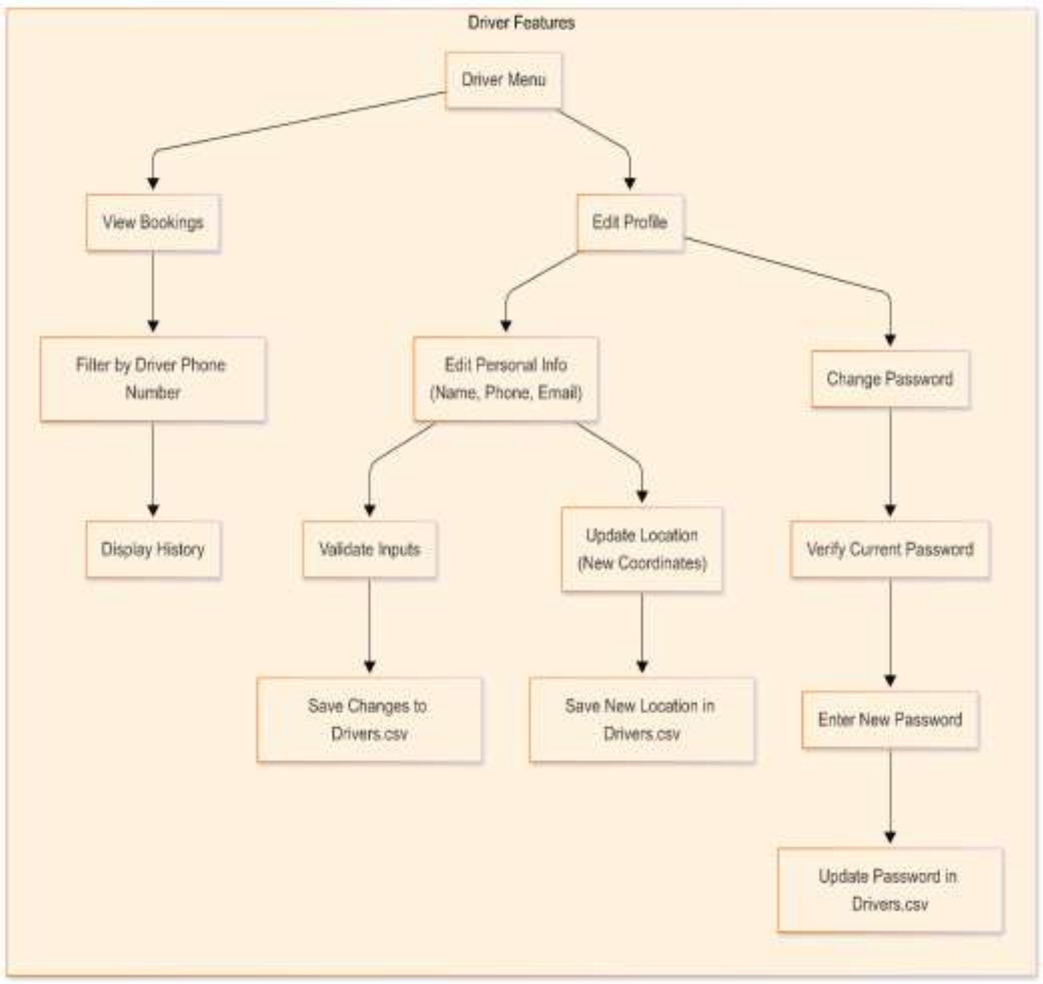
## 3) AUTHENTICATION PROCESS:



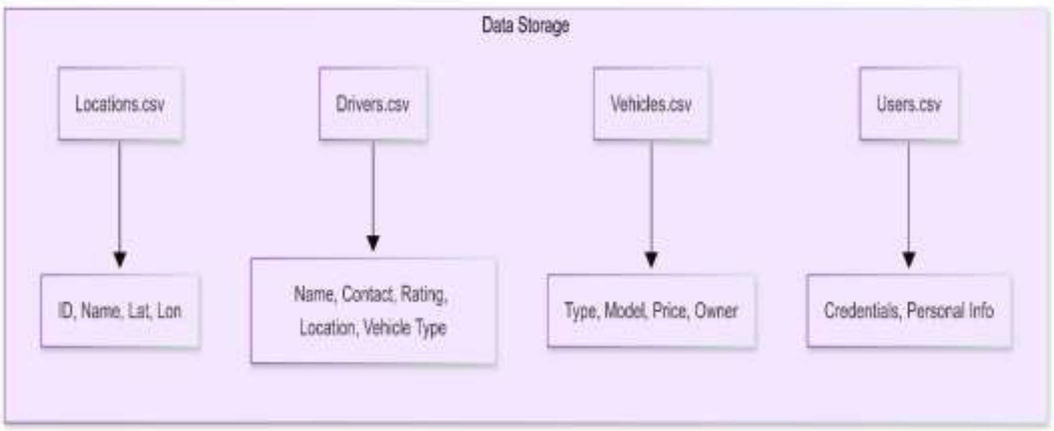
## 4) USER FEATURES :



## 5) DRIVER FEATURES :


Driver Features
Driver Menu
View Bookings
Edit Profile
Filter by Driver Phone Number
Edit Personal Info (Name, Phone, Email)
Change Password
Display History
Validate Inputs
Update Location (New Coordinates)
Verify Current Password
Save Changes to Drivers.csv
Save New Location in Drivers.csv
Enter New Password
Update Password in Drivers.csv

## 6) DATA STORAGE :


Data Storage
Locations.csv
Drivers.csv
Vehicles.csv
Users.csv
ID, Name, Lat, Lon
Name, Contact, Rating, Location, Vehicle Type
Type, Model, Price, Owner
Credentials, Personal Info

## Requirements to Accomplish This Project :

### A) Technical Requirements:

1) Programming Language: C++ (leveraging modern features from C++11 and beyond for enhanced syntax clarity, performance optimization, and better abstraction practices).
2) Compiler: Any standard-compliant C++ compiler such as GCC g++, Clang, or Microsoft Visual C++.
3) Standard Libraries: Extensive use of the C++ Standard Library including: <iostream>, <vector>, <string>, <fstream>, <sstream>, <iomanip>, <ctime>, <cmath>, <stdexcept>, <limits>, <algorithm>, <cctype>, <climits>, <cfloat>, <unordered_map>, <chrono>, <memory>, <conio.h>, and others.
4) Operating System: Platform-independent — developed and successfully tested on Windows 10/11 environments.
5) Development Environment: Visual Studio Code (with C++ extension), Code::Blocks IDE, GCC g++ compiler, and Dev-C++.

### B) Hardware Requirements:

1) Processor: Intel Core i3 (8th Gen) / AMD Ryzen 3 or better (Recommended: i5/i7 or Ryzen 5 for smoother file I/O).
2) Memory (RAM): Minimum 4 GB RAM (Recommended 8 GB or higher for seamless execution and multitasking).
3) Storage: At least 100 MB of free disk space for executable and CSV data files.
4) Display: Console-friendly — requires basic terminal or command prompt interface.
5) Other: Standard keyboard for CLI input interaction (no mouse or graphical interface needed).

### C) Conceptual Requirements:

1) Object-Oriented Programming (OOP): Core principles such as encapsulation, abstraction, inheritance, and polymorphism have been applied extensively for modular, maintainable, and scalable system design.
2) Data Structures: Usage of std::vector for dynamic storage of users, drivers, locations, and history records. Application of std::unique_ptr for rental vehicle memory management, ensuring safe dynamic allocation and automatic deallocation.

3) File Handling: Secure reading, writing, and updating of CSV files (Users.csv, Drivers.csv, Locations.csv, Vehicles.csv, BookingHistory.csv, RentalHistory.csv) using fstream, with exception handling for corrupted or missing files.

4) Memory Management: Smart pointers (std::unique_ptr) and RAII (Resource Acquisition Is Initialization) principles applied to prevent memory leaks and dangling pointers.

5) Input Validation and CLI Interaction: Generalized input validation using function templates (getValidInput<T>()), masked password entry using _getch(), and clear CLI prompts to guide user operations.

6) Error Handling: Structured try-catch blocks across all major file and user input operations. Custom exceptions like RatingException and InputException ensure graceful handling of invalid scenarios.

7) Mathematical Computation: Haversine formula applied for precise distance computation between geographical coordinates for realistic fare calculations.

## D) OBJECT-ORIENTED PROGRAMMING (OOPs) CONCEPTS UTILIZED:

### 1) Encapsulation:

i) Encapsulation is achieved by logically grouping related attributes and behaviors within secure class structures. Direct access to data members is avoided, and public member functions are provided to interact with private/protected data.

ii) Examples:

a) The RentalVehicle class encapsulates all vehicle-specific details like type, modelAndColor, licensePlate, ownerName, contact, and pricePerDay. These are accessed externally only through getter functions like getType(), getModelAndColor(), ensuring data hiding.

b) Similarly, in the Booking class, sensitive trip details such as driverPhone and price are protected and accessed only through methods like getPhone() or calculatePrice().

iii) This design prevents accidental or unauthorized modifications to critical attributes, ensuring data integrity across operations like rentals and bookings.

2) **Abstraction:**

   i) Abstraction is used to expose only essential features and hide internal complexities from the users.

   ii) Examples:

      a) The entire login system abstracts complex password validation, role identification, masked input reading, and CSV file searching inside the loginUser() and registerNewUser() functions. From the user's perspective, it is a simple "Enter Username" and "Enter Password" process.

      b) In cab booking (bookCab() function), users select pickup and drop points without seeing the internal calculations like Haversine formula, driver filtering based on 1km radius, price calculation logic, or booking history file updates — all those steps are abstracted internally.

   iii) Thus, users interact with a simple and intuitive interface while the system handles intricate backend operations without exposing them.

3) **Inheritance:**

   i) Inheritance is employed to avoid redundancy and model real-world hierarchies efficiently.

   ii) Examples:

      a) The RentalVehicle base class defines generic attributes (e.g., model, owner, price) and behaviors (e.g., displayDetails()).

      b) Specialized vehicle types — Motorcycle, Hatchback, Sedan, and SUV — inherit from RentalVehicle and initialize their unique type during construction.

      c) Similarly, the Booking class is inherited by TuktukBooking and TaxiBooking, where each subclass overrides price calculation logic based on their respective fare systems.

   iii) By using inheritance, the code remains DRY (Don't Repeat Yourself), and maintenance becomes simpler when adding future vehicle types or booking categories.

**4) Polymorphism:**

i) Both compile-time and runtime polymorphism are effectively utilized to achieve dynamic behavior.

ii) Examples:

a) Runtime Polymorphism:

- Booking declares a virtual method calculatePrice().
- TuktukBooking and TaxiBooking override calculatePrice() to implement different fare formulas:
    - TuktukBooking: Base fare ₹20 + ₹20/km after 0.5 km,
    - TaxiBooking: Base fare ₹100 + ₹70/km after 1 km.
- During cab booking, based on the selected vehicle type, the overridden method is automatically invoked, ensuring correct price calculation dynamically at runtime.

b) Compile-Time Polymorphism:

- Function template getValidInput<T>() allowsvalidation for multiple data types (int, double, etc.) at compile time without rewriting separate functions for each type.

iii) Thus, polymorphism brings flexibility and extensibility to the system design.

**5) Constructors and Destructors:**

i) Constructors and destructors manage object lifecycles, ensuring consistent initialization and safe memory handling.

ii) Examples:

a) Parameterized Constructors:

- RentalVehicle and all its child classes (Motorcycle, SUV, etc.) initialize their attributes through constructors at object creation.
- Booking and its subclasses (TuktukBooking, TaxiBooking) calculate initial trip prices immediately during object instantiation based on distance.

- Exception classes like RatingException are initialized with custom error messages via their constructors.

b) Defaulted Destructors:

- In classes where dynamic resource management is necessary (e.g., smart pointers with unique_ptr<RentalVehicle>), destructors ensure safe cleanup automatically following RAII principles.

iii) Proper use of constructors and destructors guarantees that every object exists in a fully valid state throughout its lifetime.

**6) Templates:**

i) Templates are leveraged to create reusable, generic solutions, minimizing redundant code.

ii) Examples:

a) template<typename T> getValidInput(const string& prompt, T minVal, T maxVal):

- This function accepts inputs of any type (integer, floating-point) and validates them against a defined range.
- It streamlines user input across the application, whether entering location IDs, ride days, or driver ratings.

b) Without templates, separate functions would have been needed for int, double, etc., making the code bulkier.

iii) Templates thus promote DRY principles, enhance type safety, and simplify maintenance.

## 7) Exception Handling:

**i)** Robust exception handling ensures system reliability and graceful error recovery.

ii) Examples:

a) Custom Exceptions:

- RatingException is thrown if users provide invalid driver ratings outside the permissible 0–5 range.
- InputException handles invalid or unexpected user inputs during registration and bookings.

b) Standard Exception Handling:

- try-catch blocks are used in critical operations like file reading (ifstream) and CSV parsing.
  E.g., if Locations.csv fails to open or parse, the program catches the exception and issues a user-friendly warning instead of crashing.

c) Secure Password Management:

- If password input fails validation (missing uppercase, lowercase, number, or insufficient length), detailed error messages are thrown, guiding users toward correction.

Thus, exception handling provides a strong safety net, ensuring the system remains user-friendly even when unexpected errors occur.

## CONCLUSION :

***"Innovation is seeing what everybody has seen and thinking what nobody has thought."***
***— Dr. Albert Szent-Györgyi***

The Advanced Cab Booking and Vehicle Rental System, built using modern C++ and grounded in Object-Oriented Programming principles, represents more than just a functional console application—it is a well-structured response to real-world mobility challenges tailored specifically for Silchar, the cultural and academic hub of Assam and home to NIT Silchar.

By weaving together secure login systems, dynamic driver management, personalized user experiences, flexible ride options, and robust offline operation, the project showcases the power of thoughtful, localized innovation. It addresses the mobility challenges of Silchar not through high-end technology alone, but through human-centered design that emphasizes accessibility, security, and empowerment.

**Highlights of the System Include:**

- Context-Aware Design: Realistic modeling of driver and passenger locations, utilizing the Haversine formula to simulate geographic accuracy.
- User Personalization and Role Management: Secure authentication, editable profiles, and tailored experiences for both users and drivers.
- Resilient Architecture: Exception-handling strategies and modular coding practices ensuring system robustness even under failure conditions.

**Looking Beyond Today:**

- Security Enhancements: Implementation of password encryption, Aadhaar data protection, and stronger privacy models.
- Graphical Interfaces: Migration towards intuitive GUI systems through frameworks like Qt, enhancing visual accessibility.
- Database Integration: Transition to scalable relational databases such as MySQL for handling larger datasets and real-time operations.
- Real-Time Services: Incorporation of live GPS tracking, driver availability prediction algorithms, and smart fare estimation.

- New Economic Models: Introduction of in-app payments, loyalty programs, and subscription-based models to sustain and scale services.

The journey from idea to execution reflects a broader truth: technology must not only innovate but also inspire. Through small but deliberate steps, systems like this can reshape the daily experiences of communities, bridging service gaps with solutions born of local understanding and global vision.

***"Technology, when harnessed with empathy, becomes an unstoppable force for good."***
***— Bryan Johnson***