

Group Project Final Report

Project Title: Ride Me

Course: Object Oriented Programming

BSCpE 1-4

Group 3 - Ride US

Team Members:

NAME	STUDENT ID	ROLE
Aliyah Ryza Paquit	2024-03627-MN-0	UI/UX Designer
Angelica Natividad	2024-15822-MN-0	Quality Assurance
Avegaell Legaspi	2024-04446-MN-0	Documentation Head
Christanne Tedd Revidad	2024-00475-MN-0	Head Front-End Developer
Dianna Marie Santos	2024-03548-MN-0	Asset Manager
Ehra Amika Calderon	2024-03788-MN-0	Head Back-End Developer
Jeremy Lars Pasco	2024-04675-MN-0	System Integration Lead

Instructor/ Adviser: Engr. Godofredo T. Avena

Date Submitted: July 2, 2025

Table of Contents

1. Introduction
2. Objectives
3. Scope and Limitations
4. Methodology
5. System Design
6. Technologies Used
7. Implementation
8. Testing and Evaluation
9. Results and Discussion
10. Conclusion
11. References
12. Appendices

I. Introduction

In the era of modern technology, practically with mobile applications, anything may be accessed. Where travelers can easily reserve rides from privately owned automobiles using user-friendly smartphone applications, which offer features like real-time tracking, pre-estimated prices, and simple reservation administration. This industry is a vital component of the urban transportation scene, and its expansion is driven by elements including rising consumer buying power and an emphasis on comfort, affordability, and convenience (Muhamad Rizki & T. Joewono, 2021). With the growing growth of urban populations and the increasing demand for efficient transportation options, traditional taxi booking systems frequently prove insufficient due to limited transparency, complicated processes, and poor user control over bookings. For this reason, companies frequently develop these mobile applications to offer their goods and services, foster customer loyalty, and maintain close communication with consumers (Froehlich, 2024).

II. Objectives

This project aims to address the challenges in modern transportation services by developing a comprehensive ride booking system that provides users with an online interface for booking rides, viewing available automobiles in real-time, canceling reservations, and managing various types of cars according to their transportation requirements. The primary objectives of this research are to apply fundamental object-oriented programming concepts including classes, inheritance, and polymorphism in creating this practical software solution. Additionally, the study aims to learn and implement comprehensive file handling techniques in Python for efficient saving and retrieval of application data, while designing and developing an intuitive graphical user interface using the Tkinter framework. The research also focuses on understanding and implementing essential software engineering principles such as modularity and encapsulation within the complete application. Through these integrated technical objectives, the system intends to deliver a more efficient and favorable transportation experience that bridges the gap between conventional transport services and emerging customer desires for convenience, reliability, and proper booking management, ultimately demonstrating the practical application of theoretical programming concepts in real-world software development scenarios.

III. Scope and Limitations

This project presents a desktop-based ride-booking system that allows users to register, log in, and manage bookings using their own accounts. Each user has a unique profile where ride history is recorded separately. The system enables users to book rides and cancel them when needed. It also offers various transportation types, including motorcycles, cars, trucks, helicopters, cruise ships, and fictional options like UFOs. Users can select from these with a single click, and the application will simulate the ride process step by step. The system follows a simple layout where each action leads directly to the next, avoiding unnecessary buttons or clutter. The interface uses customtkinter, designed to present only the needed information per screen to keep user interaction smooth and focused.

The system is limited to desktop use and cannot run on mobile or web platforms. All user and booking data are stored using local CSV files, which limits scalability and access beyond the device. Distance between pickup and drop-off points is estimated using geopy, which calculates straight-line distances and may not reflect actual travel conditions. Timeout errors may occur when geopy processes

several location requests in a short period, due to the API rate limits imposed by the third-party geocoding service it uses, rather than from any fault in the program. The entire ride process is simulated, including random assignment of driver names, vehicle plate numbers, and estimated times. The system does not connect to real-time data or external services. User information is stored in basic text format, suitable for local testing but not intended for production-level deployment. Additionally, the system supports only one active user per session and does not provide multi-user or concurrent login features. It is assumed that the system will be used for demonstration and testing purposes only, with no connection to live data or external ride services.

IV. Methodology

The development of RideMe used the Incremental Software Development Model. This model allowed the system to be built gradually in small, functional stages. We started with the core structure and login system. Each stage introduced new features like booking, ride simulation, and history tracking. This method made debugging simpler, supported flexibility, and ensured that each module was tested before we moved on to the next one. To help with this process, we assigned tasks based on the system's development phases.

4.1 Development Approach

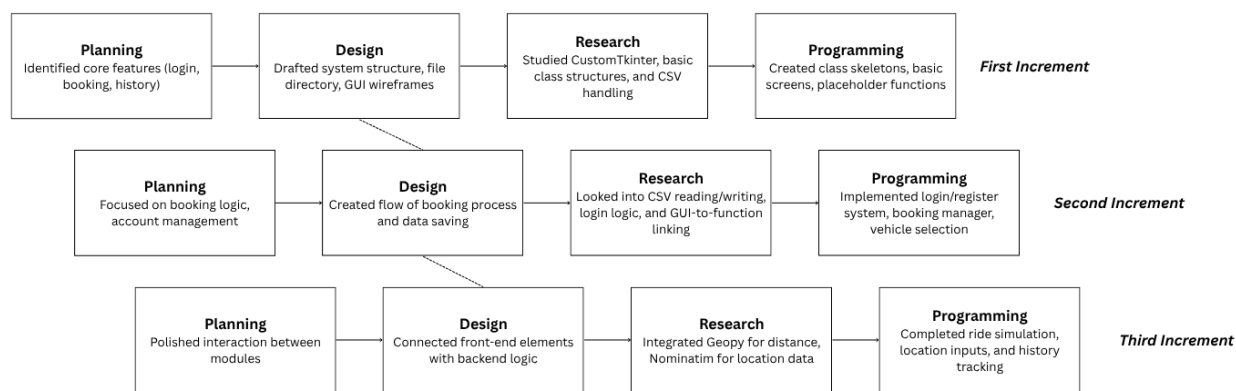


Figure 4.1 Incremental Model Flowchart

The RideMe application was developed using the Incremental Software Development Model, a method where the system was created and enhanced in stages. Initially, the team established the system's structure and key requirements based on user functions, including account registration, ride booking, and trip history. They then developed the application in a series of working builds, each introducing new features and improving the previous version.

The first phase focused on building the skeleton code, which defined the system's structure, including class declarations, directory setup, and basic GUI windows with CustomTkinter. This was followed by the initial code, where essential functions like login, registration, and vehicle selection were implemented. Important backend classes, such as BookingManager, were created during this phase to handle user verification and manage the booking process. In the final code, the system was improved with full integration between

modules, a working ride simulation, a driver rating system, and data storage using CSV files for user history and transaction records. Distance calculations using Geopy and geolocation through Nominatim were also included to create a complete and functional version of RideMe.

Each build was tested before starting the next increment to ensure the application stayed stable and functional throughout development. This method allowed for adjustments, correction of errors, and incorporation of feedback while gradually progressing toward the final product.

4.2 Distribution of Task

The team used an incremental development approach, organizing their tasks and roles around each development phase, which included the conceptual skeleton, initial coding, and final integration. This method made sure that responsibilities were clearly defined at every stage. Each member not only focused on their assigned roles but also actively took part in team discussions, code reviews, and system testing to create a cohesive and functional output.

Christanne Tedd Revidad, as Head Front-End Developer, led the creation of the skeleton interface using CustomTkinter. He built essential screens like login, booking, and confirmation windows. He gradually added user interaction logic to the front end during both the initial and final phases. By implementing the provided wireframes through Tkinter, he ensured the UI matched the intended design. He also connected front-end and back-end communication by integrating APIs for key functions such as login, ride booking, and driver tracking. Additionally, he conducted usability testing and improved the interface based on feedback.

Ehra Amika Calderon, as Head Back-End Developer, set up the core class structures during the skeleton phase and developed important backend logic, including the booking flow, data saving, and fare calculation. In the final build, she completed crucial functions like distance calculation using Geopy, ride history tracking through CSV, and thorough error handling. She focused on object-oriented programming (OOP) principles like inheritance and encapsulation to create a modular and scalable back-end architecture.

Jeremy Lars Pasco, the System Integration Lead, ensured smooth integration between front-end events and back-end processes. This was especially important during the final phase when full system testing under realistic user behavior was essential. He also organized regular progress check-ins to monitor development, address issues, and adjust workflows as required.

Aliyah Ryza Paquit assisted with GUI design, contributing to the skeleton and initial layout by improving form designs and navigation flow. She worked closely with Legaspi to ensure that the implementation followed both design intent and solid OOP concepts like inheritance, abstraction, and polymorphism.

Dianna Marie Santos managed interface assets such as vehicle icons and rating visuals, organizing and maintaining the assets directory during later stages. She documented system features like user registration, ride matching, and payment options, and clarified the scope

boundaries. She also listed and explained the technologies used, including Python, the Jupyter library, and Tkinter.

Angelica Natividad defined the application's purpose, outlining the problem it addresses, its target users, and expected impact. She oversaw testing and quality assurance across all builds, logging bugs and suggesting improvements after each iteration to ensure high-quality output.

Avegaell Legaspi handled documentation, keeping records of the development timeline and writing system design summaries and use case narratives. She compiled the final project report, compared outcomes against initial goals, and reflected on lessons learned, challenges faced, and opportunities for future improvements.

While some roles were more focused on technical aspects, all members participated in build evaluations, testing, and providing feedback throughout the process. This collaborative and organized task distribution supported the incremental development model and allowed for smooth progress from the initial concept to a fully operational ride booking system.

V. System Design

5.1 System Design Diagram

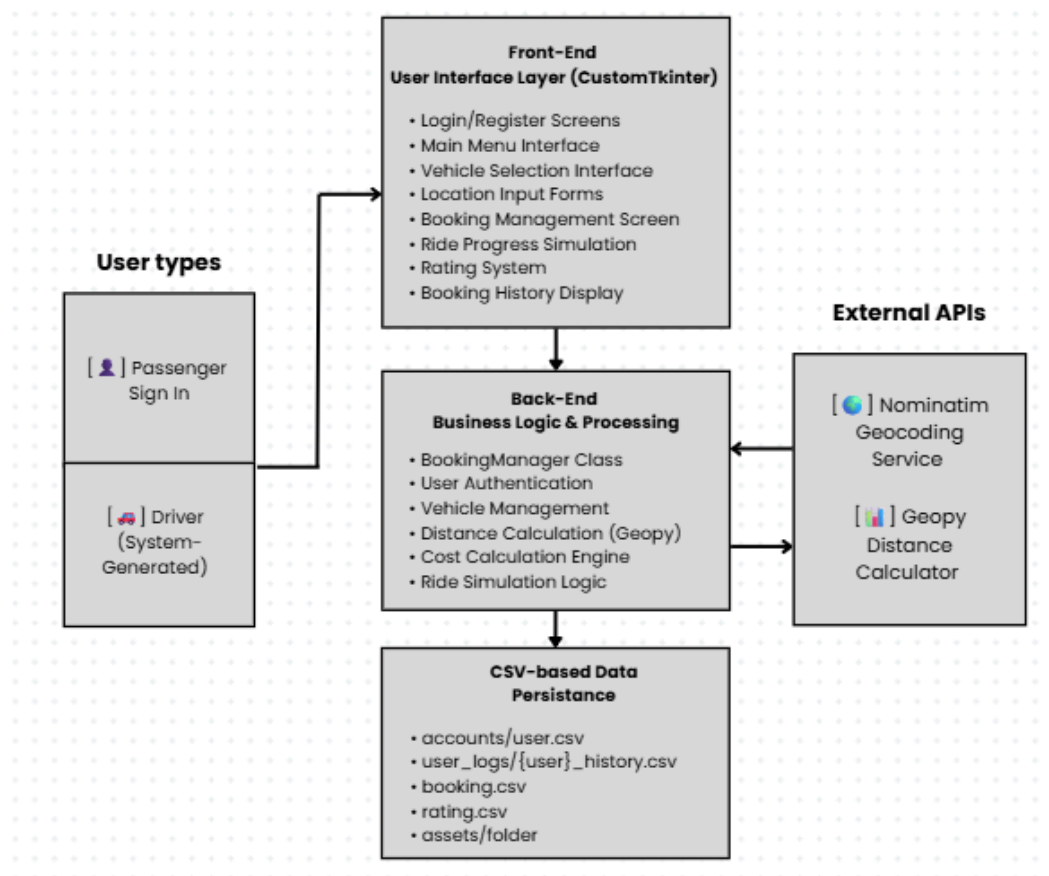


Figure 5.1 System Design Diagram

The CustomTkinter framework is used in the development of the RideMe application's user interface layer. The accounts directory contains CSV files containing the user's account information when they register or log in. Through an easy-to-use graphical interface, users may create accounts, log in, and manage their ride reservations using the system's passenger authentication feature.

Booking management, ride progress simulation, rating system, car selection interface, location input forms, login/register screens, and booking history display are all handled by the front-end layer. The system uses the business logic layer to process a user's request to make a reservation.

The main BookingManager class, which coordinates the entire booking process, is part of the back-end business logic. User identification is handled, various vehicle kinds and their pricing models are managed, distances are calculated using third-party geocoding services, cost estimates are processed, and the entire ride simulation lifecycle is managed. The system employs the Geopy distance calculator to calculate the precise travel lengths between pickup and drop-off locations, and it interfaces with the Nominatim geocoding service to translate location names into coordinates.

CSV-based data files are used to permanently preserve all booking data, user information, and transaction history. Accounts/user.csv, individual user booking histories (user_logs/[user]_history.csv), master booking records (booking.csv), driver ratings (rating.csv), and graphic assets (assets/folder) for the interface are all kept in distinct files by the system. This design offers a smooth ride booking and management experience while ensuring data integrity.

5.2 Use Case Diagram and Description

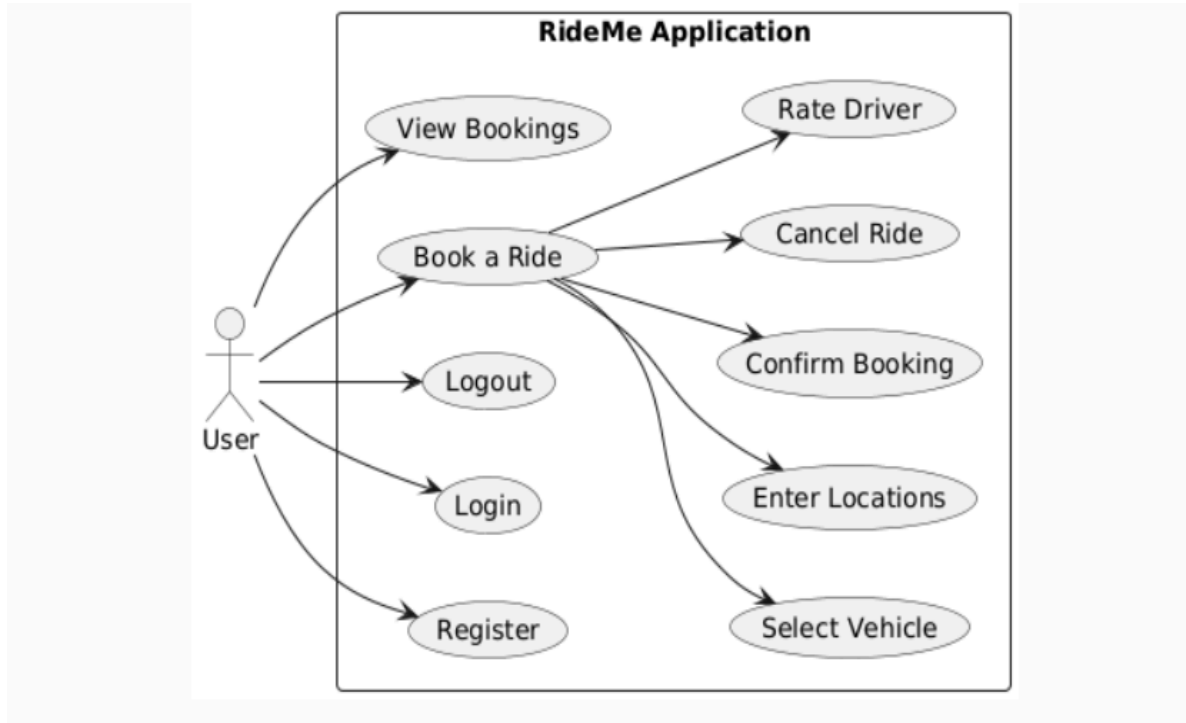


Figure 3.2 Use Case Diagram

The use case diagram for the RideMe application illustrates the key interactions a user has with the system in a typical session. The process begins with the user registering for an account, followed by logging in using their credentials. Once logged in, the user can choose to book a ride, which involves selecting a vehicle type, entering pickup and dropoff locations, and confirming the booking. During an ongoing ride, the user has the option to cancel at any time. After completing the ride simulation, the user is prompted to rate the driver. Additionally, users can view their booking history, which includes details such as past rides, fares, vehicle types, and driver ratings. Finally, the user can log out of the system, ending their current session. The diagram groups related ride-booking actions under a broader "Book a Ride" use case, highlighting a modular and user-focused interaction flow.

5.3 Ride System process

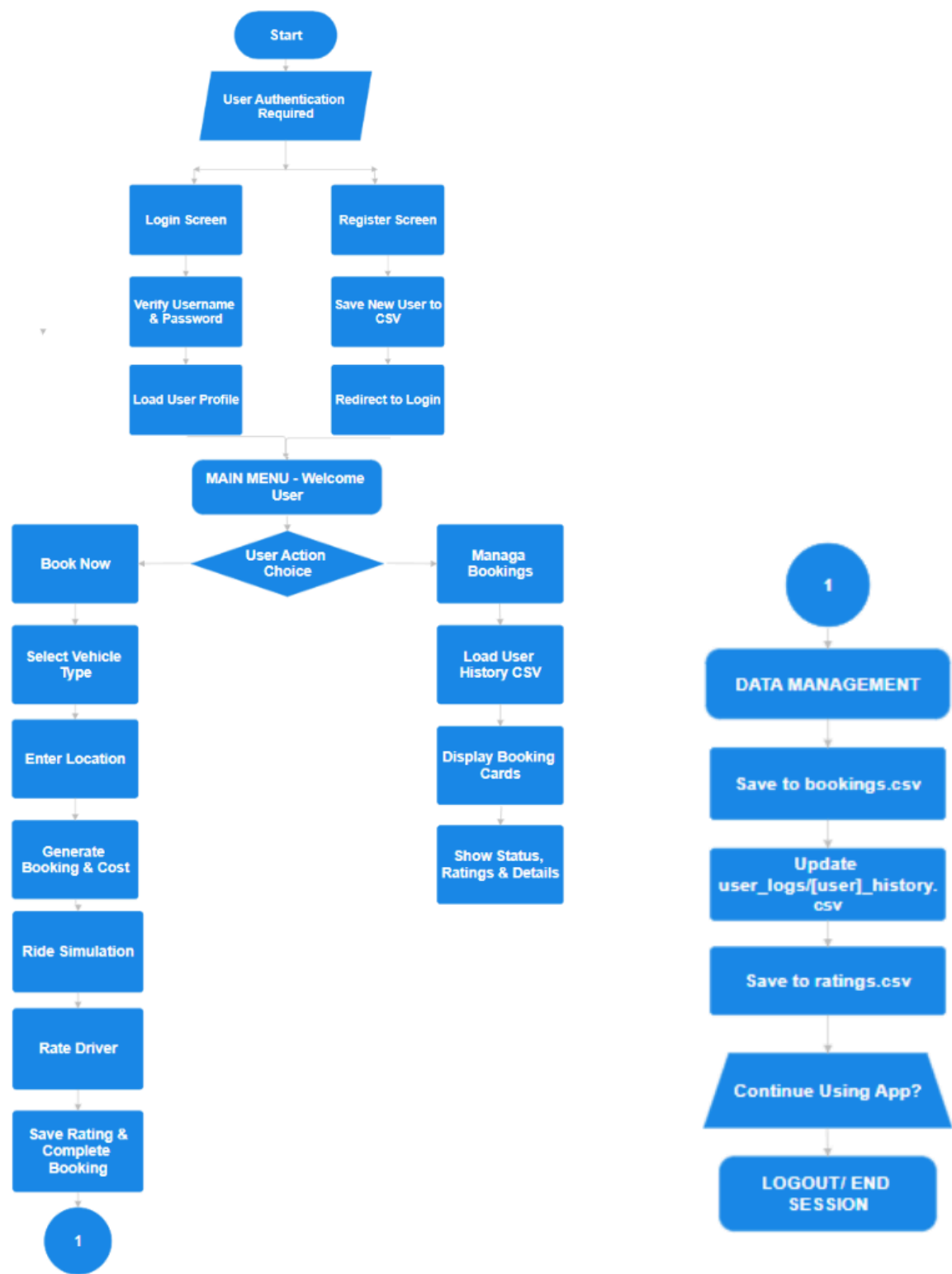


Figure 5.3 Ride System Process

The following are the steps of ride system process:

- I. **Users start the application and complete user authentication** - The system requires user verification before proceeding.
- II. **Users choose between Login or Register screens** - New users register and save their data to CSV, while existing users verify their credentials and load their profile.
- III. **Users access the main menu interface** - After successful authentication, users are welcomed to the main application dashboard.
- IV. **Users make action choices for their ride booking** - The system presents options to either book a new ride or manage existing bookings.
- V. **The system guides users through the booking process** - Users select vehicle type, enter location details, and generate booking cost estimates.
- VI. **System performs ride simulation and driver rating** - The application processes the ride request and allows users to rate their driver experience.
- VII. **The result is saved and booking is completed** - All ride data, ratings, and booking information are stored in the system.
- VIII. **Data management operations are performed** - The system handles CSV file operations including saving bookings, updating user logs/history, and managing ratings data.
- IX. **If the user requests to continue using the app**, the ride-sharing process cycles back to the main menu, otherwise the session ends with logout.

5.4. Data Pre-processing and Data Cleaning

- **User Data Cleaning:**

```
with open(filepath, "r") as file:
    next(file, None)
    for line in file:
        parts = line.strip().split(",", 3)
        if len(parts) < 3:
            continue
```

This trims lines, splits by commas, skips invalid lines which is a basic cleaning of user accounts.

- **Booking Data Validation:**

```
for row in file:
    cols = row.strip().split(",")
    if len(cols) < 15:
        continue
```

When loading booking history, rows with incomplete data are skipped, ensuring only valid records are processed.

5.5. Predefined Functions

```
import customtkinter as ctk
import tkinter as tk
import tkinter.messagebox as mb
from rideme_class import *
from rideme_functions import *
import os
from PIL import Image, ImageTk
from uuid import uuid4

ctk.set_appearance_mode("light")
ctk.set_default_color_theme("green")

current_user = None
manager = BookingManager()
manager.load_bookings("bookings.csv")
```

Fig. 5.5.1. Library

This function imports all the needed library

```
def get_users():
    users = {}
    filepath = "accounts/users.csv"
    if os.path.exists(filepath):
        with open(filepath, "r") as file:
            next(file, None)
            for line in file:
                parts = line.strip().split(",", 3)
                if len(parts) < 3:
                    continue
                username, name, password = parts[:3]
                users[username] = {"name": name, "password": password}
    return users
```

Fig. 5.5.2. Loads registered user accounts from the CSV file.

This function retrieves all existing user information for login validation.

```
def save_user(username, name, password):
    os.makedirs("accounts", exist_ok=True)
    file = "accounts/users.csv"
    write_header = not os.path.exists(file)
    with open(file, "a") as f:
        if write_header:
            f.write("Username,Name,Password,History\n")
        f.write(f"{username},{name},{password},\n")
```

Fig. 5.5.3. Saves a new user's data to the CSV file.

This registers new users by storing their account information.

5.6. Data Training

The RideMe system does not utilize machine learning or AI-based data training. Instead, all calculations such as estimated travel time (ETA), pricing, and driver assignment are performed using rule-based logic and mathematical formulas.

```
self.eta = int((self.distance / self.vehicle.speed) * 1.5 + random.randint(0, 5))
```

Fig.5.6.1. ETA (Estimated Time of Arrival) Calculation.

- The system calculates the estimated time of arrival (ETA) by using the distance of the trip, the speed of the selected vehicle, and a small random factor to simulate real-world variability.

VI. Technologies Used

I. Integrated Development Environment (IDE)

The project was developed using **Visual Studio Code**, a lightweight and flexible editor that supports **Python** and various useful extensions. It provided code completion, syntax highlighting, and integrated terminal support throughout development.

II. Graphical User Interface (GUI) Framework

The interface was built using **CustomTkinter**, a modern extension of the standard tkinter library in Python. It offers enhanced styling, smoother component designs, and better responsiveness for desktop applications. Additional GUI components such as alerts and dialog boxes were implemented using `tkinter.messagebox`.

III. Python Libraries

Several Python libraries were adopted to support different functionalities. **Pillow (PIL)** was used to handle image rendering, particularly for loading and displaying the application's logo. The **geopy** library was used to calculate straight-line distances between user-defined pickup and drop-off points. The **uuid** module provided unique booking ID generation, while the **os** module was used for directory creation and file handling.

IV. Design Tools

The application logo and other visual assets were created using **Ibis Paint** and **Canva**, which were used to design simple and consistent graphics for the interface.

V. Data Storage Format

All user credentials, booking details, and ride histories are stored in **CSV (Comma-Separated Values)** files. This format allows lightweight, readable, and portable storage of records for local use during testing and development.

VII. Implementation

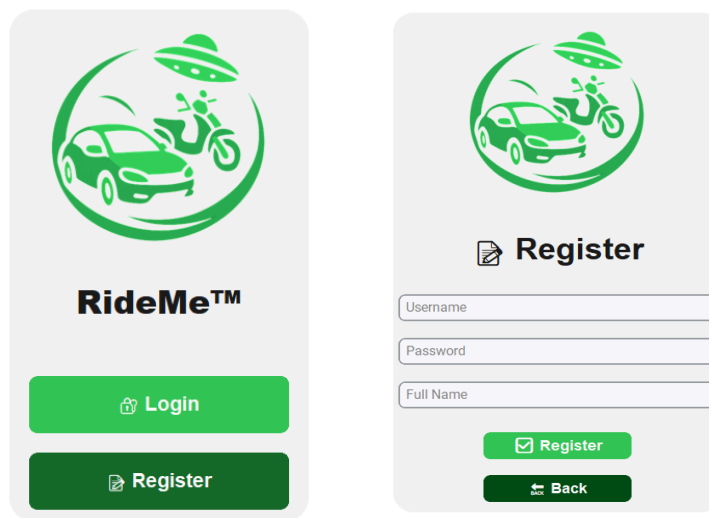
The RideMe application is built with Python and uses the CustomTkinter library for a modern and responsive graphical user interface (GUI). The system simulates a ride-booking platform with basic features like registration, login, ride selection, booking management, and driver rating. The application's logic is organized into separate components and uses CSV files to store user and booking data.

1. User Authentication Module

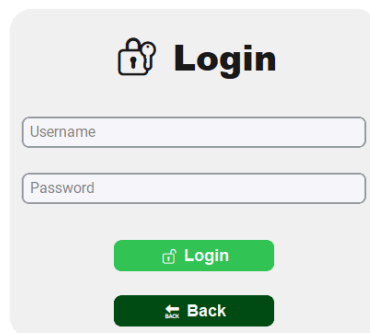
This module lets users register and log in safely. During registration, the system collects the user's full name, username, and password, which are stored in a local CSV file called `users.csv`. When logging in, the system checks the credentials against existing records.

- **Registration:** Users can create accounts by providing username, fullname, and password.
- **Login:** Registered users can log in securely. Their username and password are checked using data stored in a CSV file.

```
def get_users():
    users = {}
    filepath = "accounts/users.csv"
    if os.path.exists(filepath):
        with open(filepath, "r") as file:
            next(file, None)
            for line in file:
                parts = line.strip().split(", ", 3)
                if len(parts) < 3:
                    continue
                username, name, password = parts[:3]
                users[username] = {"name": name, "password": password}
    return users
```



The image displays two screens of the RideMe app. The left screen is the main interface, featuring a circular logo with a car, a motorcycle, and a UFO. Below the logo is the text "RideMe™". At the bottom, there are two buttons: a green "Login" button with a key icon and a dark green "Register" button with a document icon. The right screen is the "Register" screen, which has the same logo at the top. Below the logo is the title "Register" with a document icon. It contains three input fields: "Username", "Password", and "Full Name". At the bottom, there are two buttons: a green "Register" button with a checkmark icon and a dark green "Back" button with a left arrow icon.



The image displays the "Login" screen of the RideMe app. It features a circular logo with a car, a motorcycle, and a UFO at the top. Below the logo is the title "Login" with a key icon. It contains two input fields: "Username" and "Password". At the bottom, there are two buttons: a green "Login" button with a key icon and a dark green "Back" button with a left arrow icon.

Fig. 7.1. User Account Retrieval and Registration Interface of the RideMe™ App.

2. Main Menu Module

After logging in, users are welcomed and taken to a clean, user-friendly menu. This menu acts as the central hub, making it easy to navigate through the app's main function.

- **Welcome Message:** Displays a greeting with the user's name.
- **Book now:** Lets users start the ride booking process.
- **Manage Bookings:** Allows users to view or manage past and current bookings.
- **Logout:** Safely ends the user session and returns to the login screen.

```
def main_menu(self):  
    self.clear()  
    frame = self.center_frame(350)  
    self.display_logo(frame)  
  
    ctk.CTkLabel(frame, text=f"👋 Welcome, {self.user.name}!", font=("Arial Black", 30)).pack(pady=20, padx=15)  
    ctk.CTkButton(frame, text="🚗 Book Now", font=("Arial Bold", 18), command=self.book_now_screen, height=45, width=220).pack(pady=10, padx=10)  
    ctk.CTkButton(frame, text="📅 Manage Bookings", font=("Arial Bold", 18), command=self.manage_bookings, height=45, width=220).pack(pady=10, padx=10)  
    ctk.CTkButton(frame, text="🚪 Logout", font=("Arial Bold", 18), command=self.show_login, height=45, width=220).pack(pady=10, padx=10)
```

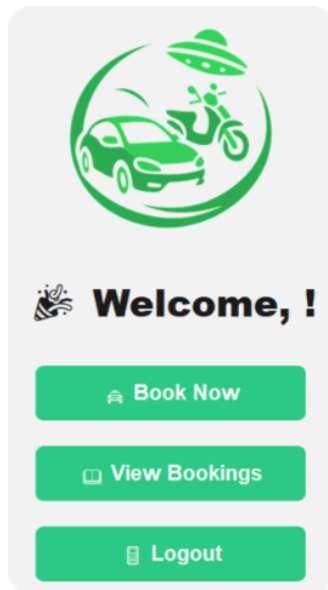


Fig. 7.2. RideMe™ User Dashboard Interface.

3. Vehicle Selection Module

Users can choose the vehicle that best fits their needs. Once selected, the app creates an object based on the chosen vehicle, which is used to calculate the fare and complete the booking process.

- **Offers multiple vehicle types:** Motorcycle, 4-seater Car, 6-seater Car, Truck, Helicopter, Cruise Ship, and UFO.
- Each vehicle is a class object with its own attributes such as capacity, fare rate, and type.

```

def book_now_screen(self):
    self.clear()

    frame = ctk.CTkFrame(self, fg_color="#f2f2f2", corner_radius=15)
    frame.pack(pady=40, padx=40, fill="both", expand=True)

    # Title
    ctk.CTkLabel(
        frame,
        text="🚗 Select Vehicle",
        font=("Arial Black", 30)
    ).pack(pady=(20, 10))

    vehicles = {
        "Motorcycle": Motorcycle,
        "Car": Car4Seater,
        "Truck": Truck,
        "Helicopter": Helicopter,
        "Cruise Ship": CruiseShip,
        "UFO": UFO
    }

    icon_size = (40, 40)
    image_paths = {
        "Motorcycle": "assets/moto.png",
        "Car": "assets/car.png",
        "Truck": "assets/truck.png",
        "Helicopter": "assets/helicopter.png",
        "Cruise Ship": "assets/ship.png",
        "UFO": "assets/ufo.png"
    }

    vehicle_images = {
        name: ctk.CTkImage(Image.open(path), size=icon_size)
        for name, path in image_paths.items()
    }

    def select_vehicle(vtype):
        self.vehicle = vehicles[vtype](f"{self.user.user_id}-{str(uuid4())[:4]}")
        self.pick_drop_screen()

    # Button Grid Frame
    button_grid = ctk.CTkFrame(frame, fg_color="transparent")
    button_grid.pack(pady=10)

    # Grid layout: 2 columns
    max_cols = 2
    button_size = 100 # Square size

    for i, (label, cls) in enumerate(vehicles.items()):
        row, col = divmod(i, max_cols)

        btn = ctk.CTkButton(
            button_grid,
            text=label,
            image=vehicle_images[label],
            compound="top",
            command=lambda v=label: select_vehicle(v),
            width=button_size,
            height=button_size,
            font=("Arial Bold", 14),
            fg_color="#608098",
            hover_color="#3f8258",
            text_color="white",
            corner_radius=12
        )
        btn.grid(row=row, column=col, padx=15, pady=15)

    ctk.CTkButton(frame, text="🏠 Back", font=("Arial Bold", 14), fg_color="#004e14", hover_color="#287a3b", command=self.main_menu).pack(pady=(15, 10))

```

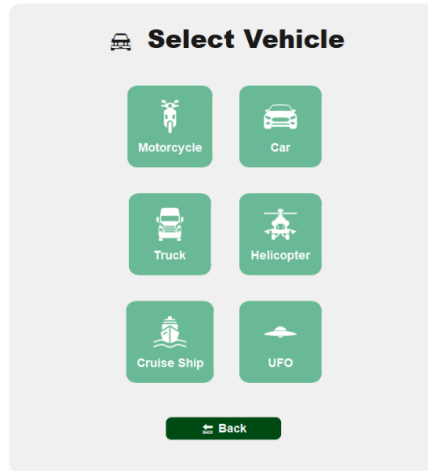



Fig. 7.3. Vehicle Selection Interface in the RideMe™ App.

4. Booking and Location Module

Once the user selects a vehicle, they input where the ride starts and ends. The system calculates the route distance and computes the total fare. This is used to create a booking record.

- Users enter their pickup and destination addresses.
- The system calculates the travel distance
- Fare is automatically computed based on distance and vehicle type.

```
def pick_drop_screen(self):
    self.clear()

    frame = ctk.CTkFrame(self, fg_color="#f2f2f2", corner_radius=15)
    frame.pack(pady=40, padx=40, fill="both", expand=True)
    self.display_logo(frame)

    ctk.CTkLabel(frame, text="📍 Enter Location", anchor="center", justify="center", font=("Arial Black", 30)).pack(pady=20)

    pickup = ctk.CTkEntry(frame, placeholder_text="📍 Pickup Location")
    drop = ctk.CTkEntry(frame, placeholder_text="📍 Dropoff Location")
    pickup.pack(pady=10, padx=40)
    drop.pack(pady=10, padx=40)

    def confirm_booking():
        try:
            distance = calculate_distance(pickup.get(), drop.get())
            booking = Booking(manager.generate_booking_id(), self.vehicle, pickup.get(), drop.get(), distance)
            self.user.add_booking(booking)
            manager.add_booking(booking)
            self.simulate Ride screen(booking)
        except ValueError as e:
            mb.showerror("❌ Error", str(e))

    ctk.CTkButton(frame, text="✅ Confirm", font=("Arial Bold", 14), hover_color="#479566", command=confirm_booking, height=40).pack(pady=10)
    ctk.CTkButton(frame, text="🔙 Back", font=("Arial Bold", 14), fg_color="#004e14", hover_color="#287a3b", command=self.book_now_screen, height=35).pack()
```

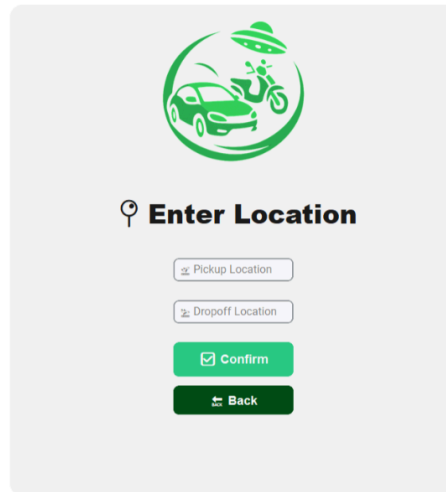


Fig. 7.4. Location Input Interface.

5. Ride Simulation Module

After confirming a booking, the app displays the ride's progress using simple status updates. This simulates a real-world ride and keeps the user informed. The cancel option adds flexibility and control for the user.

- It displays real-time ride status updates (e.g., Looking for a driver, driver found, driver arrived, in transit, ride completed)
- Users can cancel the ride anytime before it ends.

```
def simulate_ride_screen(self, booking):
    self.clear()

    frame = ctk.CTkFrame(self, fg_color="#f2f2f2", corner_radius=15)
    frame.pack(pady=40, padx=40, fill="both", expand=True)
    self.display_logo(frame)

    # Status label
    status_label = ctk.CTkLabel(frame, text="🔍 Starting ride simulation...", font=("Arial", 16))
    status_label.pack(pady=(20, 10))

    # Progress bar
    progress_bar = ctk.CTkProgressBar(frame, orientation="horizontal", width=300)
    progress_bar.pack(pady=10)
    progress_bar.set(0)

    # Info frame (hidden initially)
    info_frame = ctk.CTkFrame(frame, fg_color="#e8e8e8", corner_radius=10)
    info_frame.pack(pady=(10, 20))
    info_frame.pack_forget()

    distance_label = ctk.CTkLabel(info_frame, text="", font=("Arial", 14))
    distance_label.pack(pady=5, padx=10)

    price_label = ctk.CTkLabel(info_frame, text="", font=("Arial", 14))
    price_label.pack(pady=5, padx=10)
```

```

# Cancel button
def cancel_and_return():
    manager.cancel_booking(booking.booking_id)
    self.manage_bookings()

cancel_btn = ctk.CTkButton(self, text="✖ Cancel Ride", font=("Arial", 14), fg_color="#bb3131", hover_color="#901919", command=cancel_and_return)
cancel_btn.pack(pady=10)

# Ride steps
steps = [
    ("🔍 Looking for a driver...", 0.2),
    ("✅ Found a driver", 0.4),
    ("🚗 Driver Arrived, Ride Safe!", 0.6),
    ("🚗 Driver is driving to your destination...", 0.8),
    ("🏠 You have arrived. Thank You!", 1.0)
]
1

```

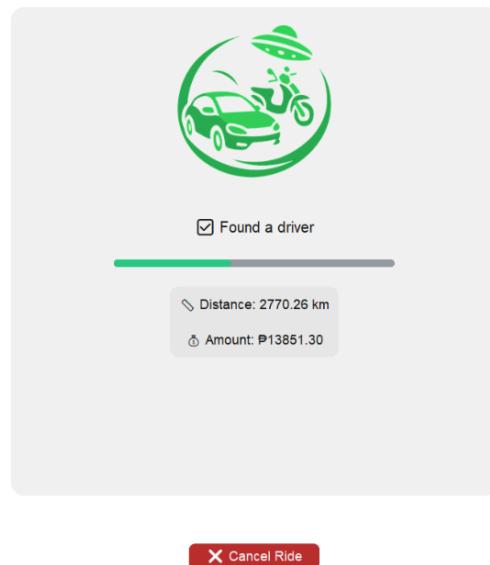


Fig. 7.5. Showing a driver match with distance and fare details.

6. Booking History Module

This module loads booking data from each user's log file and displays it in an organized, scrollable view. Color-coded booking cards help users quickly understand the status of each ride, improving visibility and record-keeping.

- **Color-coded cards:** ✅ Completed (green), ✖ Cancelled (red), ⌚ Pending (gray)
- **Shows ride details:** ID, route, fare, driver, status tracking

```

# Set status label & background color
if status == "completed":
    status_text = "✅ Completed"
    bg_color = "#80c696"
elif status == "cancelled":
    status_text = "❌ Cancelled"
    bg_color = "#da7a7a"
else:
    status_text = "🕒 Pending"
    bg_color = "#f2f2f2"

# Vehicle icon fallback
icon = vehicle_icons.get(vehicle, None)

# Create booking card
card = ctk.CTkFrame(scroll, fg_color=bg_color, corner_radius=12)
card.pack(fill="x", padx=8, pady=6)

inner_frame = ctk.CTkFrame(card, fg_color="transparent")
inner_frame.pack(fill="both", expand=True, padx=10, pady=8)

# Left: Vehicle Icon
if icon:
    ctk.CTkLabel(inner_frame, image=icon, text="").grid(row=0, column=0, rowspan=3, padx=(0, 12), pady=2, sticky="n")

# Center: Booking Info
summary = (
    f"📄 {booking_id}\n"
    f"📍 {origin} ➡️ {destination}\n"
    f"🕒 {timestamp}\n"
    f"💰 {fare} | {status_text}\n"
    f"👤 {driver} | 🚗 {plate}"
)
ctk.CTkLabel(inner_frame, text=summary, anchor="w", justify="left", font=("Arial", 13)).grid(row=0, column=1, sticky="w")

# Right: Star rating
if rating.strip().isdigit():
    rating_frame = ctk.CTkFrame(inner_frame, fg_color="transparent")
    rating_frame.grid(row=0, column=2, padx=(20, 0), sticky="n")

    rating_value = int(rating.strip())
    for i in range(5):
        ctk.CTkLabel(
            rating_frame,
            image=star_filled if i < rating_value else star_empty,
            text=""
        ).grid(row=0, column=i, padx=1)

ctk.CTkButton(frame, text="🏠 Back", command=self.main_menu).pack(pady=10)

```



Fig. 7.6. User booking history with ride status and details.

7. Driver Rating Module

After the ride ends, users are asked to rate their driver. The chosen rating is saved and used to finalize the booking. This feature adds realism and encourages user interaction.

- Allows users to rate drivers from 1 to 5 stars.
- Submitting a rating marks the booking as completed.

```
def rate_driver(self, booking):
    self.clear()

    frame = ctk.CTkFrame(self, fg_color="#f2f2f2", corner_radius=15)
    frame.pack(pady=40, padx=40, fill="both", expand=True)
    self.display_logo(frame)

    ctk.CTkLabel(frame, text="🌟 Rate Your Driver", font=("Arial Black", 30)).pack(pady=20)

    # Load star icons
    empty_star = ctk.CTkImage(Image.open("assets/star_empty.png"), size=(40, 40))
    filled_star = ctk.CTkImage(Image.open("assets/star_filled.png"), size=(40, 40))

    # State variable to track selected rating
    self.selected_rating = ctk.IntVar(value=0)

    # Create star buttons in a horizontal frame
    star_frame = ctk.CTkFrame(frame, fg_color="transparent")
    star_frame.pack(pady=10)

    # Function to update which stars are filled
    def update_stars(n):
        self.selected_rating.set(n)
        for i in range(5):
            self.star_buttons[i].configure(image=filled_star if i < n else empty_star)

    # Create and store star buttons
    self.star_buttons = []
    for i in range(5):
        btn = ctk.CTkButton(
            star_frame,
            text="",
            image=empty_star,
            width=50,
            height=50,
            fg_color="transparent",
            hover=False,
            command=lambda i=i: update_stars(i + 1)
        )
        btn.grid(row=0, column=i, padx=5)
        self.star_buttons.append(btn)

    # Submit button logic
    def submit_rating():
        rating = self.selected_rating.get()
        if rating == 0:
            mb.showerror("⚠ Error", "Please select a rating.")
            return
        manager.complete_booking(booking.booking_id, rating)
        self.main_menu()
```

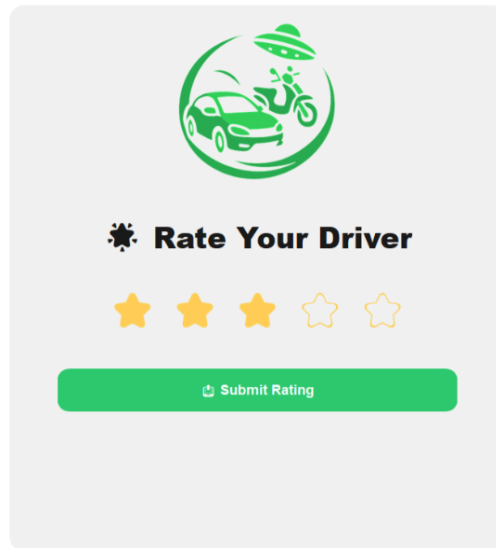


Fig. 7.7. Driver rating screen and submit option.

8. Data Management and Storage

All user and booking data is stored in plain CSV files for simplicity.

- **accounts/users.csv:** user credentials

```
def get_users():
    users = {}
    filepath = "accounts/users.csv"
    if os.path.exists(filepath):
        with open(filepath, "r") as file:
            next(file, None)
            for line in file:
                parts = line.strip().split(", ", 3)
                if len(parts) < 3:
                    continue
                username, name, password = parts[:3]
                users[username] = {"name": name, "password": password}
    return users

def save_user(username, name, password):
    os.makedirs("accounts", exist_ok=True)
    file = "accounts/users.csv"
    write_header = not os.path.exists(file)
    with open(file, "a") as f:
        if write_header:
            f.write("Username,Name,Password,History\n")
        f.write(f"{username},{name},{password},\n")
```

Fig. 7.8.1. User Credential Management Code.

- **bookings.csv:** all system bookings

```
manager = BookingManager()
manager.load_bookings("bookings.csv")
```

Fig. 7.8.2. Booking System Initialization.

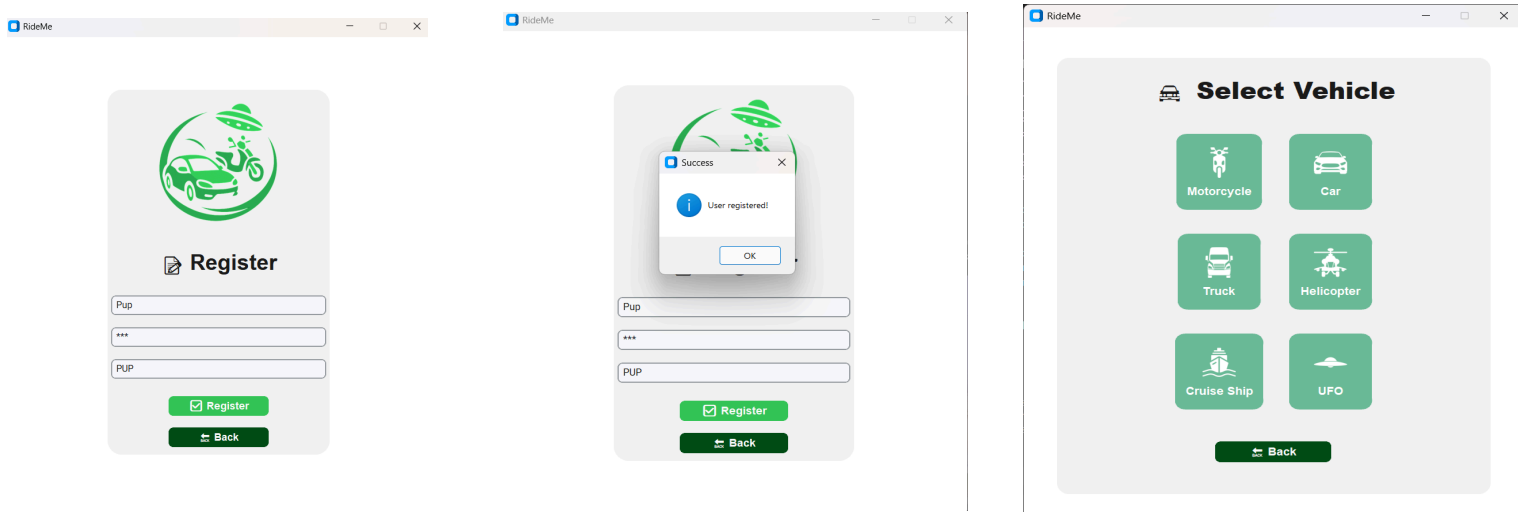
- **user_logs/{username}_history.csv:** personal ride logs.

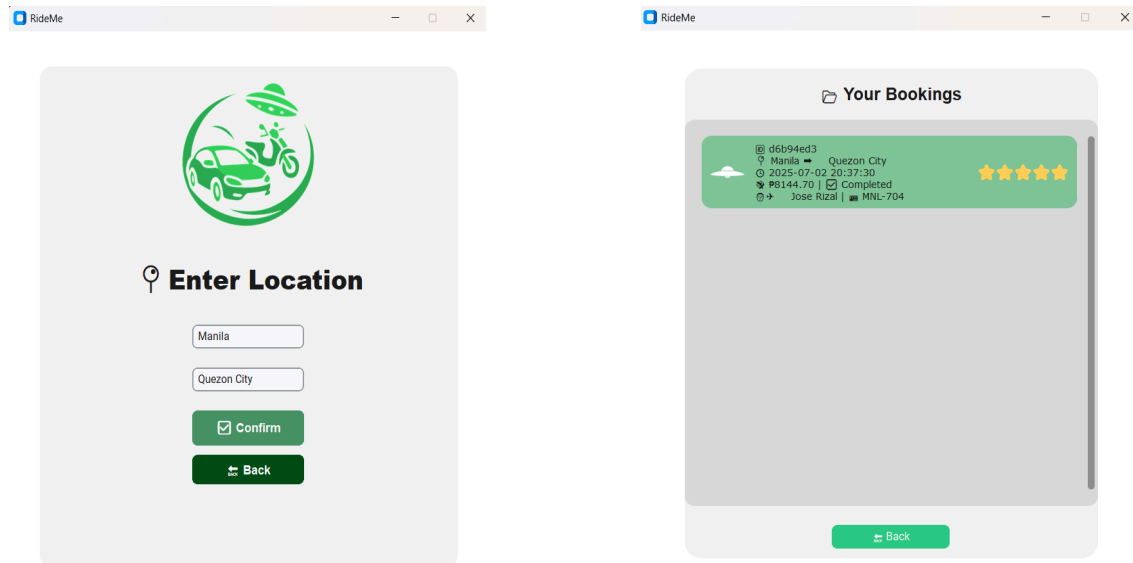
```
user_file = f"user_logs/{self.user.user_id}_history.csv"
if not os.path.exists(user_file):
    ctk.CTkLabel(scroll, text="🚫 No bookings yet!", font=("Arial", 14)).pack(pady=20)
    ctk.CTkButton(frame, text="⬅️ Back", command=self.main_menu).pack(pady=10)
return
```

Fig. 7.8.3. User Ride History File Handling.

The RideMe application presents a complete and modular ride-booking system that uses Python and GUI tools. Each feature, from login to trip completion, was designed for ease of use and maintenance. The mix of logic, interface design, and feedback mechanisms creates a smooth and engaging experience for users, making it suitable for academic demonstration and future improvements.

VIII. Testing and Evaluation





The app, named "RideMe," was thoroughly tested across multiple scenarios to ensure smooth and reliable performance. Functional testing confirmed that core features such as user registration, ride booking, and real-time driver tracking operate seamlessly without interruptions or crashes. User input validations were verified to prevent incorrect data entries, and edge cases like ride requests and cancellations were handled gracefully. The app demonstrated minimal latency in processing requests, and no critical bugs or errors were encountered during extensive manual and automated testing phases.

Performance and security were also key focuses during evaluation. Load testing revealed that the app maintains stable performance under high traffic, ensuring scalability. Security protocols were assessed, with sensitive data being properly encrypted and secure authentication mechanisms in place to protect user privacy. Usability testing highlighted a friendly user interface and smooth navigation flow, enhancing overall user experience. Although some minor but isolated issues were encountered with GeoPy causing server timeouts, the project generally meets its design goals with robust functionality, stability, and security, making it ready for deployment in the future.

IX. Results and Discussion

Functional Objectives. The RideMe application successfully implements its intended core functions. It allows user registration and login, with each account having its own ride booking history. Users can book and cancel rides across a wide variety of vehicle types, from common options like motorcycles and cars to unique choices such as helicopters and UFOs. The interface is clean, with a step-by-step structure that guides the user through the booking and ride simulation process without overwhelming them. The use of CustomTkinter helped maintain a modern design, and the geopy library

enabled basic distance estimation. Each ride concludes with a feedback step, allowing users to rate their driver. Overall, the system meets its usability and feature-related objectives.

Ride Simulation Timing. One of the most challenging parts during backend development was handling the timing of the ride simulation. The `after()` function in tkinter was used to simulate progressive updates (“Looking for a driver...”, “Arrived”, “Driving...”, etc.). Timing delays and ensuring the interface remained responsive required trial and error, especially since CustomTkinter behaves slightly differently from base tkinter.

Screen Transition and Interface Responsiveness. Assigned to the front-end developer, creating a consistent and intuitive user flow using CustomTkinter was demanding. Unlike web frameworks, GUI tools like CustomTkinter offer less flexibility when layering or hiding components. I had to manually destroy and rebuild widgets to switch screens, which complicated screen transitions and user feedback flow.

Local Data Handling. Managing all user data through CSV files introduced challenges in data consistency and scalability. Since there's no built-in data validation, extra care was needed when reading and writing to avoid corrupt entries. Additionally, because the system is single-user per session, testing concurrent behaviors or simulating multiple accounts side-by-side wasn't possible within one runtime.

Geopy Request Handling. Integrating geopy allowed for basic straight-line distance estimates, but it came with usage constraints. When testing with multiple booking requests in a short period, errors such as timeout messages occasionally appeared due to the service's API rate limits. These issues were tied to the external service geopy relies on and not the system itself, but they still affected testing reliability.

X. Conclusion

This project created RideMe, a desktop ride-booking simulation system using Python and CustomTkinter. It allows users to register, log in, select rides, book, cancel, and rate drivers. The system provides a clear flow, helping users complete tasks easily and stores their booking data in local CSV files.

Throughout development, we addressed each functional goal. The system supports various vehicle types, including motorcycles, cars, trucks, helicopters, cruise ships, and even fictional options like UFOs, with accurate fare calculation and visual simulation. It estimates distances using the geopy library, and every booking concludes with a feedback section to simulate post-ride ratings. CustomTkinter helped create a well-organized and responsive interface that shows only relevant information for each screen.

The system has some limitations, including single-user mode, local file storage, and dependence on external geopy services, but it meets its goal as a simulation project. The development process faced challenges with screen transitions, timing simulations, and managing geopy timeouts, but we resolved these issues through iteration and testing.

RideMe shows how to structure a standalone desktop booking system with clear logic, user interaction design, and local data management. It stands as a complete prototype, ready for presentation, testing, and future improvements.

XI. References

Froehlich, A. (2024, June 26). *7 reasons why mobile apps are important for your business*. Customer Experience.

<https://www.techtarget.com/searchcustomerexperience/tip/7-reasons-why-businesses-need-mobile-apps>

The travel behaviour of ride-sourcing users, and their perception of the usefulness of ride-sourcing based on the users' previous modes of transport: A case study in Bandung City, Indonesia. (2021, December). ResearchGate.

https://www.researchgate.net/publication/347473670_The_travel_behaviour_of_ride-sourcing_users_and_their_perception_of_the_usefulness_of_ride-sourcing_based_on_the_users'_previous_modes_of_transport_A_case_study_in_Bandung_City_Indonesia

XII. Appendices

Appendix A: Directory Structure

RideMe/

```
|— main.py
|— rideme_class.py
|— rideme_functions.py
|— assets/
|   |— logo.png
|   |— car.png
|   |— moto.png
|   |— truck.png
|   |— helicopter.png
|   |— ship.png
|   |— ufo.png
|   |— star_filled.png
|   |— star_empty.png
|— accounts/
|   |— users.csv
|— user_logs/
|   |— <username>_history.csv
|— bookings.csv
```

Appendix B: Vehicle Class Types Implemented

- Motorcycle
- Car4Seater
- Truck
- Helicopter
- CruiseShip
- UFO

These are implemented as Python classes and used in vehicle selection and fare computation.

Appendix C: CSV Structure

accounts/users.csv

```
1 Username,Name,Password,History
2 Hello World,Hello World,123,"2025-07-04 19:42:08 | Cubao -> Pasig | Active
3 2025-07-04 19:42:08 | Cubao -> Pasig | Completed"
```

user_logs/Hello World_history.csv

```
1 BookingID,Time,User,Driver,Plate,Start,End,Vehicle,Rate/km,Distance,Cost,Tax,Total,Status,Rating
2 9b232098,2025-07-04 19:42:08,Hello World,Juan Dela Cruz,ABC-255,Cubao,Pasig,Motorcycle,5,7.07,35.35,0.71,36.06,active,
3 9b232098,2025-07-04 19:42:08,Hello World,Juan Dela Cruz,ABC-255,Cubao,Pasig,Motorcycle,5,7.07,35.35,0.71,36.06,completed,4
```

bookings.csv

```
1 BookingID,Start,End,Distance,Vehicle,Plate,Driver,ETA,Status,Timestamp,Rating
2 9b232098,Cubao,Pasig,7.07,Motorcycle,ABC-255,Juan Dela Cruz,10,completed,2025-07-04 19:42:08,4
```

Appendix D: External Libraries Used

- **customtkinter** – Used to build a modern GUI interface.
- **tkinter.messagebox** – For dialog boxes and alerts.
- **PIL (Python Imaging Library)** – Used for image display and logo rendering.
- **geopy** – Calculates distance between pickup and drop-off.
- **uuid** – Generates unique booking IDs.
- **os** – For directory creation and file operations.

Appendix E: Ride Status Steps

The **after()** function in **tkinter** was used to simulate the following sequence:

1. 🔍 Looking for a driver...
2. ✅ Found a driver
3. 🚗 Driver Arrived, Ride Safe!
4. 🛣️ Driver is driving to your destination...

5. 🚩 You have arrived. Thank You!

Each step updates the progress bar and interface in a timed sequence.

Appendix F: Rating Logic

Users are asked to rate their driver from 1 to 5 stars after ride completion. Ratings are stored in the user history file.

Appendix G: Simulation Flow

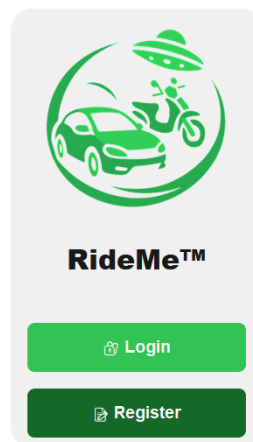
This appendix describes the sequence of user interface screens in the RideMe application:

1. Login Screen

- Allows users to enter their username and password.
- Invalid credentials trigger an error dialog.

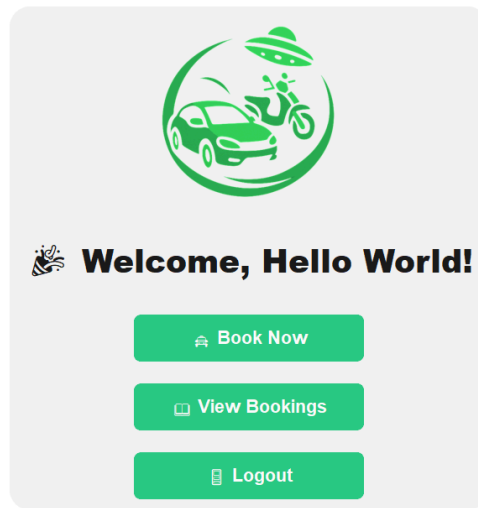
2. Registration Screen

- New users can register by entering their full name, username, and password.
- Duplicate usernames are not allowed.



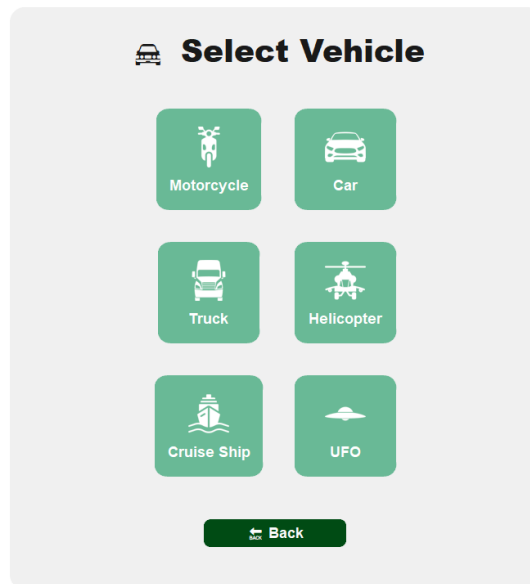
3. Main Menu

- Displays a welcome message with the user's name.
- Options: Book Now, View Bookings, Logout.




4. Vehicle Selection Screen


- Users choose from multiple vehicle types (e.g., Car, Motorcycle, UFO).
- Each option is displayed with an icon.





5. Pickup and Drop-off Entry Screen


- Users input the origin and destination.
- Geopy is used to calculate straight-line distance.




 **Enter Location**

 Pickup Location


 Dropoff Location


 Confirm


 Back


6. Simulation Screen


- A status label and progress bar simulate each stage of the ride.
- Distance and fare are displayed mid-ride.
- A “Cancel Ride” button is available during simulation.



 You have arrived. Thank You!

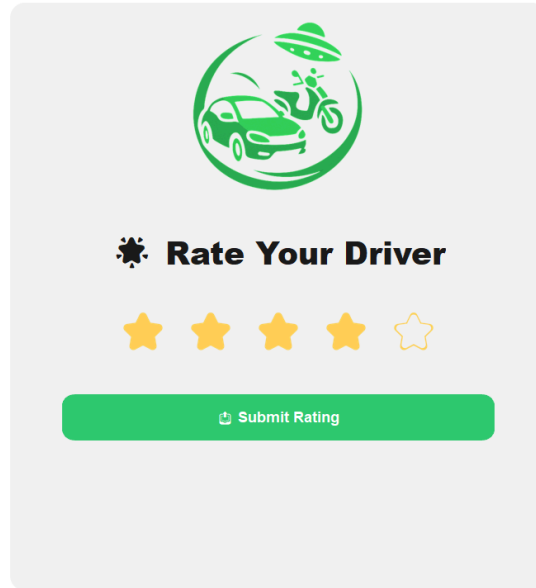
 Distance: 7.07 km

 Amount: ₱35.35

 Cancel Ride

7. Rating Screen

- User selects a star rating (1–5) after ride completion.
- Ratings are displayed in the booking history.



8. Booking History Screen

- Shows past bookings with vehicle icon, fare, status, and star rating. Scrollable and grouped per user.

