Group Project Final Report

# Project Title: Ride Booking System Project Course:

Object Oriented Programing

# Year / Section / Group Name / Number:

1 – 4 BSCPE Group #5

# Team Members:

| Name | Student ID | Role |
| --- | --- | --- |
| GUNDAYA, Miguel Gabriel S | 2024-04068-MN-0 | Project Manager |
| MAHINAY, Zaniele Adrian G. | 2024-12142-MN-0 | Lead Developer |
| MENDOA, Miguel Jose A. | 2024-03099-MN-0 | UI/UX Designer |
| VICTORIA, Julian Adam T.  VISITACION, Mark Bensent  V.  BOBILES, Zaki J.  NUYDA, Karen B. | 2024-02901-MN-0  2024-04715-MN-0  2024-05193-MN-0  2024-01683-MN-0 | QA/Test Engineer  QA/Test Engineer  Assistant Developer  Assistant Developer |

# 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 <User interface>

8. Testing and Evaluation

9. Results and Discussion (Challenges)

10. Conclusion

11. References

12. Appendices

# Introduction

This project was created to simulate a simple yet functional ride booking system that can be used on a desktop computer. It was designed with the goal of making the booking process easier for users who prefer typing in exact addresses rather than using GPS pinning or dragging map markers, which are often required in mobile ride-hailing apps. Many existing ride-booking systems focus on smartphone users and rely on location services, but that doesn’t always work well for people using desktops or laptops—especially in school environments, research labs, or areas with limited internet access.

The motivation for building this system came from the idea of making a cleaner, more accessible interface that anyone can use without needing advanced devices. By focusing on typed input, the system allows users to enter both pickup and drop-off addresses, preview them on a live map, and get a calculated fare based on distance. After confirming the locations, the user is able to choose a driver from a list of options, which adds a realistic and interactive feel to the booking process.

This system was also developed to demonstrate how Python and libraries like Tkinter, Geopy, and Google Maps Services can be used together in a practical project. It serves both as a functional ride-booking prototype and as a learning exercise in GUI programming, address geocoding, map visualization, and basic fare computation. The project aims to bridge the gap between traditional app experiences and educational tools by offering a straightforward, step-by-step interface with a more modern and visually appealing design.

# Objectives

**General Objectives:**

To develop a desktop-based ride booking system using Python that allows users to choose a vehicle, input pickup and drop-off addresses, view the route on a map, calculate estimated pricing based on distance, and select a driver, providing a user-friendly simulation of modern ride-hailing

services.

**Specific Objectives:**

* To design a graphical user interface (GUI) using Tkinter that is sleek, responsive, and intuitive for users of all levels.
* To allow users to select between different vehicle types (Motorcycle, Car, Van), each with a corresponding rate per kilometer
* To enable users to enter typed addresses for both pickup and drop-off points, improving accessibility for desktop environments without relying on GPS pinning.
* To integrate a live map display using Google Maps Integration, showing the geocoded locations and route preview on OpenStreetMap tiles.
* To calculate the estimated route distance and travel time using the OSRM API or fallback to Geopy for offline estimation.
* To compute the total fare dynamically based on selected vehicle type and route distance.
* To provide a simulated driver selection interface, allowing users to choose from a list of available drivers with visual profile icons.
* To save the booking transaction details, including vehicle type, addresses, distance, price, and selected driver, into a .csv file for record-keeping.
* To handle input validation and error messaging to ensure smooth user experience even when errors like invalid addresses occur.

# Scope and Limitations

The ride booking system developed in this project offers a user-friendly desktop-based experience that allows users to book rides through a streamlined interface. It enables users to select a vehicle type such as Motorcycle, Car, or Van, each with a predefined rate per kilometer. Users can enter specific pickup and drop-off addresses, which are geocoded and displayed on an interactive OpenStreetMap view. The system calculates the estimated distance and travel time between the two locations using the OSRM routing API. If this service is unavailable, the system automatically switches to geodesic distance estimation using Geopy.

After viewing the route and map, users proceed to a simulated driver selection interface where they choose from a set of predefined drivers, each represented with images and names. Upon confirmation, all booking details including addresses, vehicle type, route information, estimated cost, and selected driver are saved into a CSV file for reference. The graphical interface is visually enhanced for better usability, with larger windows, organized layouts, and relevant images to make the experience more intuitive.

However, the system has limitations. It does not support real-time GPS pinning or dragging markers on the map; users must manually type addresses. Driver selection is purely simulated and not based on real-time availability or actual proximity. Additionally, the system does not include advanced features like fare surges, traffic-aware route adjustments, user login, or in-app payments. All pricing is fixed and based solely on distance and vehicle type, and there is no mechanism for tracking or contacting drivers beyond the selection screen. Despite these limitations, the system serves as an effective simulation of basic ride-booking functionality for desktop environments.

# Methodology

The development of the ride booking system followed a collaborative and modular approach, ensuring that each group member had a focused role in bringing together the different components of the project. The team of six members divided the work based on individual strengths and areas of interest to ensure an efficient workflow and consistent progress throughout the development phase.

One member was responsible for designing and implementing the graphical user interface (GUI) using Tkinter. This involved setting up window layouts, managing user inputs, and ensuring a clean and responsive design. A second member focused on the mapping functionality using the Google Maps Services

module. They handled the integration of OpenStreetMap tiles and managed map updates based on user input. Another member was tasked with implementing the geolocation and routing logic. They worked on address geocoding using Geopy and distance calculation using the OSRM API, including fallback mechanisms when the primary service failed.

A fourth member handled the vehicle and pricing logic. This included defining vehicle types, applying pricing rules based on distance, and calculating estimated travel costs. The fifth member developed the driver selection interface. They created a simulated list of drivers with corresponding images and implemented the logic for driver selection and booking confirmation. Lastly, the sixth member took charge of file handling and data persistence, managing the CSV logging of all booking records and ensuring that all data from the booking process was properly saved and organized.

Throughout the project, the team maintained clear communication and regularly merged their individual contributions into a unified codebase. This collaborative structure not only helped distribute the workload evenly but also ensured that the system was developed cohesively and on schedule.

# System Design

**System Design Overview:**

The system follows an object-oriented design that organizes the booking process into modular windows:

1. Vehicle Selection
2. Address Input and Mapping
3. Driver Selection
4. Booking Summary and CSV Logging

Each stage is represented by a class, and shared logic (e.g., distance calculation, geocoding) is handled separately.

**Data Flow Summary**

VehicleSelectionWindow

↓

AddressBookingWindow

↓

DriverSelectionWindow

↓

CSV Log / Booking Confirmation

**Class Diagram**

1. VehicleSelectionWindow

Attributes:

vehicle\_var: stores selected vehicle

Methods:

go\_to\_input(): transitions to address entry window

2. AddressBookingWindow

Attributes:

vehicle: vehicle type passed from selection

pickup\_entry, dropoff\_entry: entry widgets for addresses

pickup\_coords, dropoff\_coords: coordinate tuples

Methods:

geocode\_and\_update\_map(): converts addresses to coordinates

confirm\_booking(): validates pins and proceeds to driver window

3. DriverSelectionWindow

Attributes:

vehicle, pickup\_coords, dropoff\_coords, pickup\_addr, dropoff\_addr

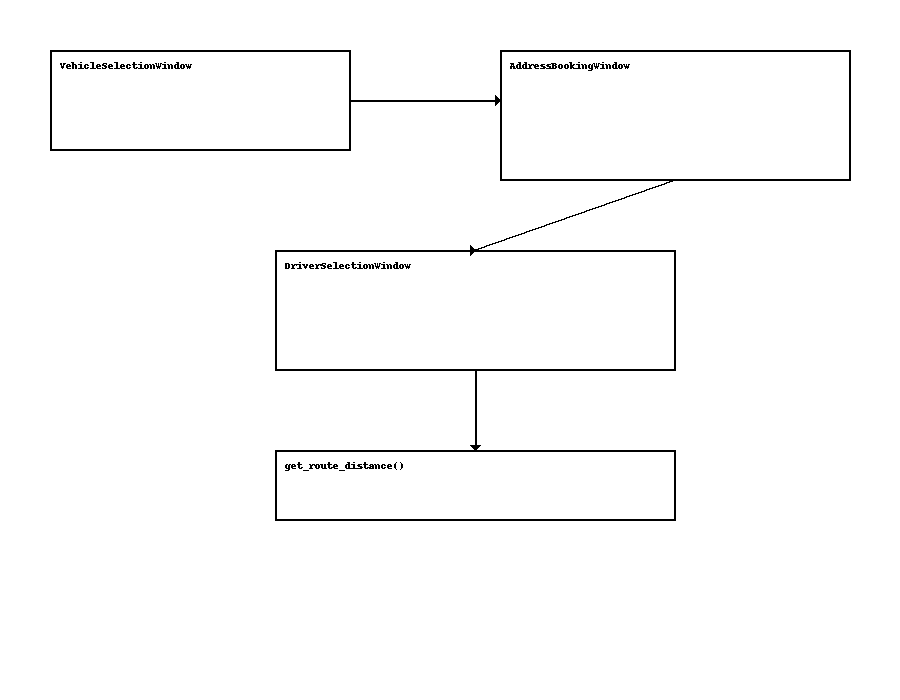
selected\_driver: stores selected driver string

Methods:

finalize\_booking(): computes distance, logs data, confirms ride

4. get\_route\_distance(pickup, dropoff)

A standalone function that returns distance, estimated duration, and method (OSRM or fallback Geopy).



# Technologies Used

**Programming Language**

* Python

Used for the entire application logic, UI, and data handling.

* Javascript, Html

Map integration

**Libraries and Modules**

* Tkinter

For creating the graphical user interface (GUI).

* Google Maps Services

For embedding and interacting with OpenStreetMap tiles in the app.

* Geopy

For converting typed addresses into geographic coordinates (geocoding).

* Requests

For fetching routing data (distance and duration) from OSRM API.

* csv (Python standard library)

For saving and reading ride booking records in CSV format.

* os (Python standard library)

For handling file paths and image loading.

**APIs and Services**

* Nominatim API (OpenStreetMap)

Used via Geopy to convert user-entered addresses into map coordinates.

* OSRM (Open Source Routing Machine)

Used to calculate route distance and estimated travel time between two points.

**Tools and Platforms**

* Jupyter Notebook

Used during development and testing of code snippets, logic blocks, and prototyping.

* Visual Studio Code (VS Code)

Used for writing, organizing, debugging, and maintaining the complete project code.

# Implementation

**1. Vehicle Selection Module**

**Description:**

This is the initial interface where the user selects the type of vehicle for their ride. Options include:

Motorcycle

Car

Van

Each option is accompanied by an image, and selection is made using radio buttons. Once the user clicks "Next", the system saves the selection and proceeds to the address entry module.

**Code Snippet:**

self.vehicle\_var = tk.StringVar(value="Car")

for v in VEHICLE\_RATES:

tk.Radiobutton(frame, text=v, variable=self.vehicle\_var, value=v).pack()

**2. Address Booking Module**

**Description:**

This module allows users to type in pickup and drop-off addresses. Upon clicking “Show on Map”, the system geocodes the addresses using Geopy, displays markers on a live map, and centers the view.

If valid, the user may proceed to the driver selection phase

**Features:**

Entry fields for location

Live map powered by Google Maps Service

Reverse geocoding fallback if internet is down

**Code Snippet:**

pickup\_location = geolocator.geocode(pickup\_address)

self.pickup\_coords = (pickup\_location.latitude, pickup\_location.longitude)

self.pickup\_marker = self.map\_widget.set\_marker(\*self.pickup\_coords, text="Pickup")

**3. Driver Selection Module**

**Description:**

Users choose a driver from a list of fictional profiles, each with:

Name

Estimated arrival tim

Image avatar

The chosen driver is included in the booking details and shown in the confirmation summary.

**Code Snippet:**

self.selected\_driver = tk.StringVar()

for driver in self.fake\_drivers:

tk.Radiobutton(frame, text=driver['name'], variable=self.selected\_driver, value=driver['name']).pack()

**4. Booking Confirmation & CSV Logging**

**Description:**

Once the driver is selected:

The system calculates route distance using the OSRM API

Fallback to Geopy if OSRM fail

It computes pricing based on vehicle type

Saves booking record into bookings.csv with:

* Vehicle type
* Locations
* Distance and duration
* Price
* Selected driver

**Code Snippet:**

with open(BOOKING\_FILE, 'a', newline='', encoding='utf-8') as f:

writer = csv.writer(f)

writer.writerow([

self.vehicle,

self.pickup\_addr,

self.dropoff\_addr,

f"{distance} km",

f"₱{price:.2f}",

self.selected\_driver

])

# Testing and Evaluation

The development team used manual testing throughout the creation of the system. Testing was conducted after each major feature was implemented, using both Jupyter Notebook and Visual Studio Code environments. The testers simulated user interactions to validate behavior across different modules, such as selecting a vehicle, entering valid and invalid addresses, visualizing the route on the map, selecting drivers, and finalizing bookings.

Each module was tested independently and then integrated to verify the end-to-end flow. Special attention was paid to data validation, UI responsiveness, and file logging behavior.

Bugs were tracked manually using a shared document among team members. Common issues that arose during testing included:

* Crashes when the user attempted to confirm a booking without setting both pickup and drop-off addresses.
* Errors caused by geocoding failures due to network delays or invalid addresses.
* UI freezes when the map tile server was slow to respond.
* Incorrect pricing due to missing or overwritten distance data.
* GUI layout inconsistencies across different screen resolutions.

| **Test Case** | **Input** | **Expected Result** | **Actual Result** |
| --- | --- | --- | --- |
| Select Vehicle | User selects "Car" and clicks Next | Address entry UI is displayed | Passed |
| Empty Address Field | Leave both address fields blank | Warning message appears | Passed |
| Valid Map Update | Enter "Manila" and "Quezon City" | Markers are placed correctly on the map | Passed |
| Invalid Location Input | Input nonsense text as addresses | Error message shown about geocoding failure | Passed |
| Drive Not Selected | Skip driver selection and click Confirm | Warning message appears | Passed |
| Full Booking Flow | Complete booking from start to finish | Bookings is saved in *bookings.csv* | Passed |
| Tile Server Slow | Simulate slow internet | Map takes longer to load but doesn't crash | Passed |

# Results and Discussion

The final ride booking system successfully met its core objectives by delivering a fully functional desktop application with a sleek and interactive user interface. Users can select a vehicle, input pickup and drop-off addresses, view the route on an embedded map, estimate the fare based on real-world distances, and choose a driver from a visually enhanced list. Each component, from geocoding and map integration to fare calculation and CSV record saving, was implemented as planned, achieving both the general and specific goals outlined at the start of the project.

Throughout development, the team encountered several challenges. One of the primary issues was ensuring accurate and responsive geolocation using typed addresses, which occasionally failed due to ambiguous inputs or API rate limits. Another technical hurdle was integrating the map view in a way that worked smoothly across all machines, as the Google Maps Services library relied on external tile servers and sometimes loaded slowly. Managing image file paths and ensuring UI consistency across different screen sizes also required additional refinement. Additionally, coordinating the contributions of six members called for strong communication and version control to avoid overwriting changes or duplicating work. Despite these challenges, the group effectively collaborated to refine and debug the system, resulting in a clean, user-centered application that simulates the ride-booking process effectively.

# Conclusion

Through the development of the ride booking system, our team gained valuable experience in applying object-oriented programming, user interface design, and real-world data integration. We learned how to work with mapping libraries, APIs, and external data sources such as OpenStreetMap and OSRM to build a functional and interactive application. The project allowed us to deepen our understanding of Python and its ecosystem while exploring how software systems interact with live services to provide location-based functionality. We also became more familiar with handling user input, file management, and displaying visual elements to create a more engaging user experience.

The impact of the project lies in its ability to simulate a simplified version of a modern ride booking platform. By allowing users to select vehicle types, input addresses, preview routes, and choose drivers, the system demonstrates how various components work together in a real-world service. It also highlights how practical solutions can be built from open-source tools, even in a desktop environment.

Most importantly, this project emphasized the value of teamwork and planning. By dividing responsibilities according to each member’s strengths, we maintained an organized workflow and ensured that every part of the system was completed efficiently. Regular communication and collaborative debugging sessions helped us solve problems faster and align our goals. The successful completion of the project reflects not only our technical progress but also our ability to work effectively as a team toward a shared outcome.

# Future Work

1. Real-time Driver Location and Assignment

Replace the simulated driver selection with real-time driver availability, where nearby drivers are shown on the map and automatically assigned based on proximity.

1. Drag-and-Drop Map Pinning

Allow users to pin their pickup and drop-off points directly on the map using mouse clicks, offering a more visual and flexible location selection.

1. User Authentication System

Add login and registration features so users can track their booking history, save favorite locations, and personalize the app.

1. Fare Estimation Details

Break down the fare with base rate, per-kilometer charge, estimated time, and optional surcharges (e.g., peak hours or high demand).

1. Booking History Panel

Include a window that displays past bookings pulled from the CSV file, with sorting and filtering options.

1. Live Traffic Data Integration

Improve travel time estimates by factoring in live traffic data from external APIs like Google Maps or HERE.

# References

OpenStreetMap contributors. (n.d.). OpenStreetMap. Retrieved from <https://www.openstreetmap.or>

OSRM Project. (n.d.). Open Source Routing Machine. Retrieved from <http://project-osrm.org>

Geopy Contributors. (n.d.). Geopy documentation. Retrieved from <https://geopy.readthedocs.io>

Python Software Foundation. (n.d.). Python documentation. Retrieved from <https://www.python.org/doc/>

TkinterMapView. (n.d.). TkinterMapView documentation. Retrieved from <https://github.com/TomSchimansky/TkinterMapView>

Requests: HTTP for Humans. (n.d.). Requests documentation. Retrieved from <https://requests.readthedocs.io>

TkDocs. (n.d.). Tkinter GUI Programming. Retrieved from <https://tkdocs.com>

Visual Studio Code. (n.d.). Code Editing. Redefined. Retrieved from <https://code.visualstudio.com>

Project Jupyter. (n.d.). Jupyter Notebook documentation. Retrieved from <https://jupyter.org>

# Appendices

**User Manual**

1. Launching the Program

Open the Python file (main.py) or the Jupyter notebook (if applicable).

Run the program. A window will appear asking you to select a vehicle.

2. Vehicle Selection

Choose one of the three vehicle types: Motorcycle, Car, or Van.

Click Next to proceed to the address input screen.

3. Enter Pickup and Drop-off Addresses

In the new window, type the pickup and drop-off addresses (e.g., "SM Mall of Asia, Pasay").

Click Show on Map to preview the route.

Make sure both addresses appear on the map with markers.

4. Confirm Booking

Once both markers are shown, click Next to proceed.

A new screen will open showing available drivers.

5. Select a Driver

Choose a driver from the list.

Click Confirm to complete the booking.

6. Booking Confirmation

A confirmation message will appear showing the details of your booking.

The booking will be saved to a file named bookings.csv.

**Installation Guide**

**Required Software:**

* Python 3.x
* Jupyter Notebook or Visual Studio Code

**Step-by-step Setup:**

1.1. Install Python packages:

pip install geopy requests

pip install webview

2.2. Ensure folder structure:

/RideBookingProject

├── images/

│ ├── motorcycle.png

│ ├── car.png

│ ├── van.png

│ ├── driver1.png

│ ├── driver2.png

│ └── driver3.png

├── bookings.csv

└── main.py

3.3. Run the App using:

python main.py

**Diagram Description**

**System Flow Diagram**

[Start]

↓

[Vehicle Selection Window]

↓

[Address Input UI]

↓

[Geocode Addresses + Show on Map]

↓

[Confirm Booking]

↓

[Driver Selection Window]

↓

[Booking Complete → Save to CSV]

↓

[Restart or Exit]

**Module Responsibilities**

| Module | Responsibility |
| --- | --- |
| VehicleSelection | Select vehicle and load images |
| AddressBooking | Input addresses and display on map |
| Geopy + OSRM | Convert address to coordinates and route |
| DriverSelection | Display drivers and confirm selection |
| Booking CSV Writer | Save booking details to bookings.csv |