

Group Project Final Report

Project Title:

Move Out: Ride Booking System

Course:

Object Oriented Programing

Year / Section / Group Name / Number:

BSCpE 1-7 - Group 7 - Woz Kids

Team Members:

<i>Name</i>	<i>Student ID</i>	<i>Role</i>
Margaret Sydney S. Laylo	2024-02317-MN-0	Project Manager
Mary Ruth P. Relator	2024-05213-MN-0	Frontend Developer
Francen Venisse E. Red	2024-01214-MN-0	Backend Developer
Marjoy M. Caranto	2024-05182-MN-0	Fullstack Developer
Marian May Legaspi	2024-03621-MN-0	Data Manager
John Reydo A. Tinawin	2024-03822-MN-0	Test Engineer / Documentation

Instructor/Adviser:

Mr. Godofredo T. Avena

Date Submitted:

July 5, 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

Move Out is a Python-based ride booking system that is designed to make every commuters' burden of finding a transport vehicle much easier. Inspired by famous ride-booking applications, Move Out also caters to commuters who need to beat the rush hour traffic and for those who want their travel to be more convenient and efficient. The system, which is supported by object oriented programming principles and basic database handling, allows users to input specific pick-up and drop-off locations, select desired vehicle type, confirm and cancel bookings, and many other functions which make the user's experience more convenient.

Existing ride-booking applications are often inconvenient, costly, and not suited to the daily transportation needs of students, many of whom struggle to secure affordable and accessible rides. These applications lack features that prioritize the financial limitations and commuting patterns of the student population. This project aims to develop a ride-booking system using the Python programming language and the Tkinter library to connect students with student-friendly transportation options, thereby promoting accessible, convenient, and economical commuting within university communities.

In addition, the primary reason for developing the system is to make a solution for the benefit of the students from Polytechnic University of the Philippines - Manila in terms of providing access to convenient transportation vehicles at an affordable cost. As a desktop-based application, the program does not fully replicate all functionalities in the mobile version and is not yet operational, but serves as a model to study on how to implement all features of a ride booking app in Python. The simple interface allows the users to navigate the system easily and explore its main functions, thus can be further enhanced in a mobile or fully integrated version in the future.

Objectives

General Objectives

To develop a ride booking system called *Move Out* that demonstrates the four pillars of object-oriented programming (OOP), including classes, inheritance, encapsulation, and polymorphism, using Python. The system aims to provide a functional and user-friendly platform for booking and managing transportation services through a graphical interface built with Tkinter. The project also seeks to implement file handling for data persistence and apply modular software design practices that reflect essential software engineering concepts in a practical setting.

Specific Objectives

1. To implement a ride booking system that supports multiple vehicle types, each with unique fare calculation and capacity rules based on real-world pricing models.
2. To develop a booking management feature that allows users to view, categorize, and track their ride history, including pending, completed, and canceled rides.
3. To ensure data persistence by storing and retrieving booking records across sessions using an SQLite database.
4. To integrate Google Maps Platform services, including the Maps Static API, Distance Matrix API, Directions API, Routes API, Geocoding API, and Places API, for features such as distance and route calculation, map display, address lookup, and location suggestions.
5. To create a help center module that provides users with accessible system instructions and frequently asked questions.

Scope and Limitations

This project involves the development of *Move Out*, a desktop-based ride-booking system created using Python, designed to demonstrate the practical application of object-oriented programming (OOP) principles. The system simulates the core features of a ride-hailing platform within a limited academic scope. It allows users to request transportation services, monitor ride status, and review booking history. Three vehicle types are available: motorcycle, four-seater car, and six-seater car, each with its own fare and capacity logic. Fare computations are based on distance and vehicle type, following rate references from the *Move It* platform.

To support location-based functionalities, the system integrates various Google Maps Platform services, including the Maps Static API, Distance Matrix API, Directions API, Routes API, Geocoding API, and Places API. These services enable features such as route generation, map display, distance and time calculation, address conversion, and

place suggestions. User interaction is handled through a graphical user interface built with Tkinter, and booking data is stored using an SQLite database to ensure persistence OF DATA across EVERY USER sessions. Additionally, the system includes a help center that provides users with manuals (if available) and frequently asked questions to support ease of use.

While the system is functional, it operates under several limitations due to its academic and prototype-based nature. First, it does not include a driver-side interface or real-time matching between drivers and passengers. Instead, ride assignments are simulated using pre-configured driver data. Although the system requires users to register with a name, phone number, and a four-digit personal identification number (PIN) that acts as a basic form of password, it does not support advanced authentication features such as email verification, password recovery, or multi-factor authentication. Moreover, while the system provides fare estimates, it does not handle actual transactions or integrate digital payment platforms.

In terms of user support, the help center is static and does not allow interactive communication, such as real-time chat or ticket-based inquiries. The system is also limited to desktop environments and is not optimized for mobile devices. Lastly, a stable internet connection is required for location-based features, as the map-related functions depend on external APIs provided by the Google Maps Platform.

Methodology

Phase 1: Planning and Delegation

Using an online meeting platform, the team members discussed first how the final project should look like, what objectives to meet, and what functions it should perform. The group then assigned respective roles for each person: Ms. Laylo as Project Manager, Ms. Relator as Frontend Developer, Ms. Red as Backend Developer, Ms. Caranto as Fullstack Developer, Ms. Legaspi as Data Manager, and Mr. Tinawin as Test Engineer and in-charge of Documentation. After the roles have been assigned, the team discussed the branding of the ride-booking system, basic features of the system, and additional features that could be added. The team also decided what apps to utilize for the entirety of the development phase.

Phase 2: Conceptualization of System Design

The design of the system was conceptualized using Figma to create wireframes and interface layouts. Design inspirations were gathered from open-source ride-booking layouts, primarily through Google Images. The focus was on creating a user-friendly layout and improving transitions between system pages for better usability. Wireframes, mockups, and the final Tkinter UI layouts were created to design the homepage, login page, sign-up page, and the system's main functions. These designs served as visual guides and structural references for the frontend development of the project ensuring consistency and alignment.

Phase 3: Program Execution

The system was developed using Visual Studio Code with Python programming language and the Tkinter library. It was divided into three main parts: frontend, backend, and database to ensure a smooth process of building the system within separated tasks. Python was used for core programming with an object-oriented structure. SQLite3 served as the local database, managed using DB Browser for SQLite. Google Maps API was integrated using the Google Maps Python Client to fetch location data and coordinates for the Map feature. Furthermore, additional libraries used included: os and sys for file and path handling, urllib.parse for API URL encoding, and Pillow for image rendering.

Overall, the GUI was built using Tkinter and connected to backend logic. All modules were tested and debugged within Visual Studio Code. Version control was maintained using GitHub, with commits pushed through Git Bash.

Phase 4: Integration of Additional Features

After building the main prototype of the system, additional features were added to improve system functionality. A Help Center function page was developed using Tkinter library widgets and Pillow for image rendering to assist users with common issues and navigation. Other enhancements were also implemented to optimize the system's accessibility, usability, and convenience for student users on other system pages.

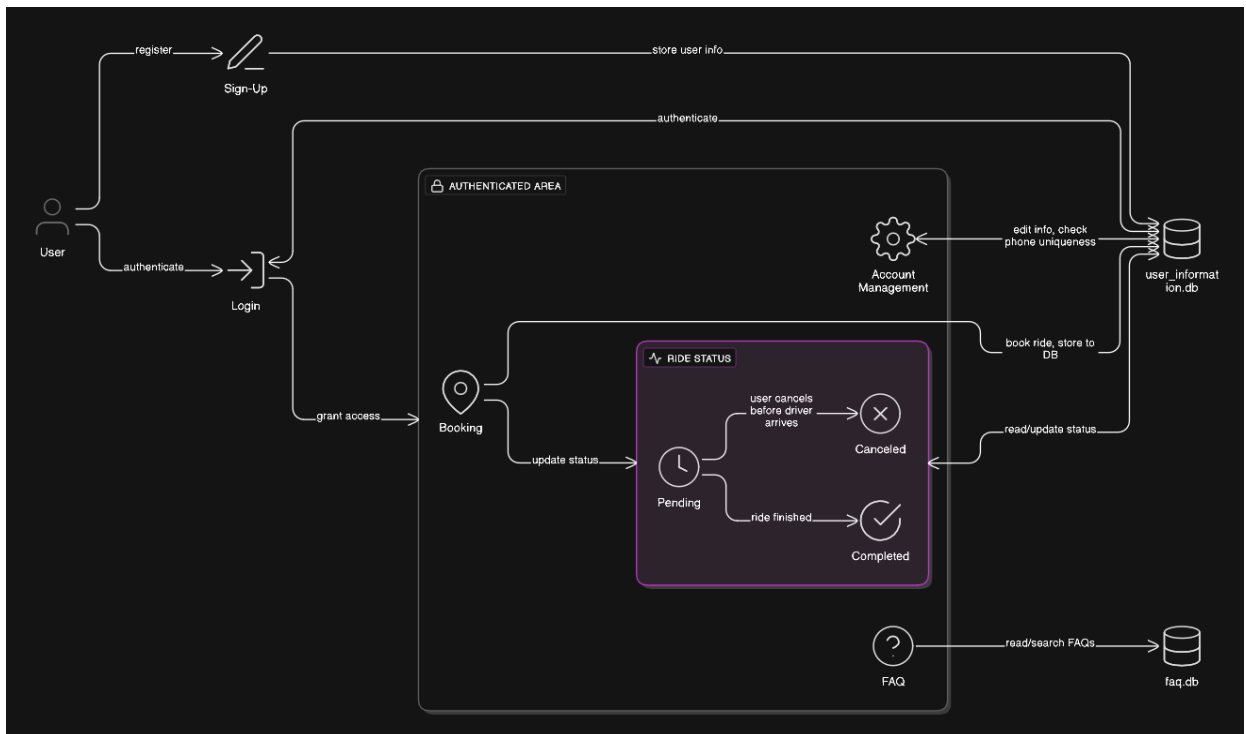
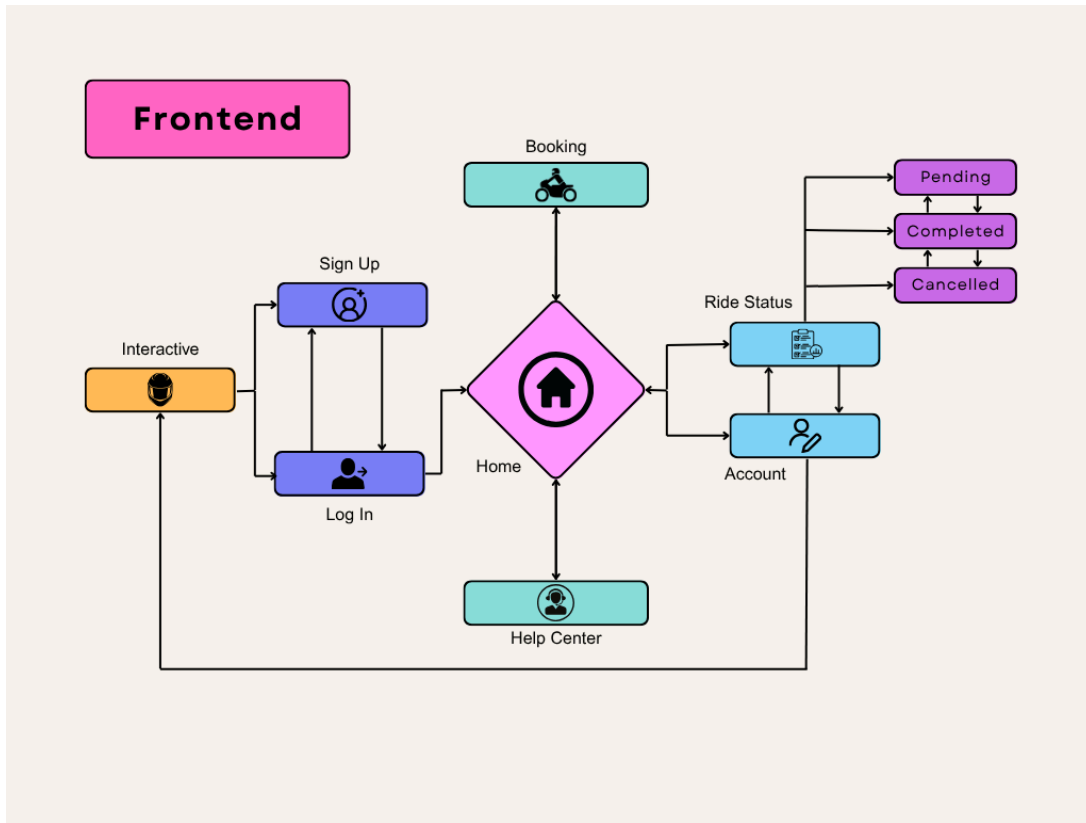
Phase 5: Testing and Review

Multiple testing methods have been used to test the app's capacity. All testing has been performed in VSCode through the terminal where the developers observed the outputs for any error. Each feature including the signup, log in, ride booking, and account editing was tested individually by putting valid and invalid data to observe if the features will respond correctly. After fixing bugs and errors, the program was tested multiple times again to see if everything remains running smoothly and no new error will occur. Bugs were tracked using VSCode terminal alongside manual taking of notes to take note of each error with proper documentation, and taking considerations of its severity level. The current state of bugs either not fixed or resolved was recorded with accountability of specified team members.

Phase 6: Finalization of the Program

After a series of rigorous testings wherein many test cases have been simulated, the program showed consistency in all its features without errors. Every test case scenario possible was verified, and the features were tagged as valid given that it runs its functions smoothly without errors. Given that all documented bugs have been resolved, the program was then finalized and is ready for publishing.

System Design



Technologies Used

- **Python**
 - For object oriented programming
- **SQLite3**
 - For database
- **Visual Studio Code**
 - IDE of the main program
- **DB Browser**
 - To show tables with its values
- **Tkinter**
 - To construct basic GUI applications
- **Figma**
 - To create simulated layout of the program
- **Canva**
 - Made layouts for figma
- **Pinterest**
 - For layout inspirations
- **Adobe After Effects**
 - For splash animation editing
- **Adobe Media Encoder**
 - Export video file from After Effects to a GIF file
- **Google Cloud Console**

APIs and Services:

 - Maps Static API
 - For embedded static map image
 - Distance Matrix API
 - To get travel time and distance between points A and B
 - Directions API
 - Directions between points A and B
 - Routes API
 - A combined version of Distance Matrix and Directions API. Used for creating a path between points A and B.
 - Geocoding API
 - For fetching location and address coordinates.
 - Places (NEW) API
 - For showing suggested places based on user input
- **Pillow**
 - Used to insert images
- **Github**
 - For repository and all folders
- **Git bash**
 - To push updates to repository
- **OS**
 - For fetching directories of files

- **sys**
 - For accessing relative file paths
- **random**
 - For generating random variables
- **urllib.parse**
 - To encode path parameters into a valid URL
- **String**
 - For generating a collection of string constants
- **Google Maps**
 - Used to interact with Google Maps API and premade functions (autocomplete, distance, coordinates, duration, static map) under Google Maps

Implementation

System Architecture Overview

The system is built on a modular architecture using **Python** with the **Tkinter** library for the graphical user interface and **SQLite3** for local data storage. The design ensures a clear separation between interface, backend logic, and database interaction.

1. Main Window as Controller

- The **Main Window** acts as the central controller, managing the initialization and navigation of all pages.
- Each screen is implemented as a separate class, with the main window providing shared access to common methods and data.
- Navigation between screens is handled programmatically through method calls in the main controller.

2. Database Layer

- The application uses **SQLite3** as the local relational database.
- It handles storage and retrieval for:
 - User account information
 - Ride booking data (including pending, completed, and canceled rides)
- Centralized database methods are used to interact with data, ensuring consistency and minimizing duplication.

3. Backend Logic

- The backend module handles all booking-related logic, including:
 - **Map and route simulation**
 - **Pickup and drop-off location suggestions**
 - **Estimated ride duration computation**
 - **Unique booking ID generation**
 - **Driver and vehicle assignment simulation**

Main Modules and Their Functions

Each major screen in the application is implemented as a separate class to ensure modularity and clear responsibility. Below are the main modules and their core functionalities:

- **Interactive Screen**

Displays an introductory animation when the app is launched.

```
...
    This module provides the interactive page for Move Out.
    It allows the user to navigate to the login and signup pages.
...

from tkinter import *
import tkinter.font as tkFont

class InteractivePage(Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.parent = parent
        self.configure(bg="#ffc4d6")
```

- **Login Page**

Allows users to log in using their registered credentials.

```
...
Login Page UI for the MOVE OUT application using Tkinter.

This module defines the LoginPage class, which provides the graphical interface for user login,
including mobile number, PIN entry, and navigation to signup.
...

from tkinter import *
from tkinter.font import Font
from tkinter import ttk

from backend.login import LoginHandler

class LoginPage(Frame):
    def __init__(self, parent):
        super().__init__(parent, bg="#ffc4d6")
        self.parent = parent
        self.login_handler = LoginHandler() # Access backend logic for user accounts
```

- **Sign-Up Page**

Enables new users to register and create an account in the system.

```
...
Sign Up Page UI for the MOVE OUT application using Tkinter.

This module defines the SignUpPage class, which provides a styled user interface
for account registration, including fields for first name, last name, mobile number,
PIN entry, and navigation back to login.
...

from tkinter import *
from tkinter.font import Font
from tkinter import ttk

from backend.signup import SignupHandler

class SignUpPage(Frame):
    def __init__(self, parent):
        super().__init__(parent, bg="#ffc4d6")
        self.parent = parent
        self.signup_handler = SignupHandler() # Access backend logic for creating new user accounts
```

- **Home Page**

Acts as the dashboard after login. Users select pickup and drop-off locations here before proceeding to booking.

```
from tkinter import *
import tkinter.font as Font

class HomePage(Frame):
    ...
    This program provides the home page for Move Out.
    It allows the user to book their chosen ride.
    ...
    def __init__(self, parent):
        super().__init__(parent)
        self.parent = parent
        self.configure(bg="white")

        self.suggestions_box = None
        self.active_entry = None
```

- **Booking Page**

Lets users choose the type of vehicle and view the calculated fare based on their selected route.

```
import sys
import os
import tkinter.font as Font

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from tkinter import *
from PIL import Image, ImageTk
from backend.ride_booking import RideBackend
from urllib.request import urlopen
from io import BytesIO

class BookingPage(Frame):
    """
    This program provides the booking page for Move Out.
    It allows the user to book their chosen ride
    and see the pinned locations in the map.
    """
    def __init__(self, parent, pickup_location, dropoff_location):
        super().__init__(parent)
        self.parent = parent
        self.configure(bg="white")

        self.pickup_location = pickup_location
        self.dropoff_location = dropoff_location
        self.distance_km = 0.0

        self.backend = RideBackend("AIzaSyAOKrot@g067ji8DpUmxN3FdXRBfMsCvRQ")
```

- **Looking for Driver Page**

Simulates the system searching for a nearby driver, with a short delay to mimic real-world processing.

```
import sys
import os
import tkinter.font as Font

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from tkinter import *
from PIL import Image, ImageTk
from backend.ride_booking import RideBackend
from urllib.request import urlopen
from io import BytesIO

class LookingPage(Frame):
    """
    Displays the "Looking for a Ride" screen.
    Shows ride details, estimated duration, and allows canceling the ride.
    """
    def __init__(self, parent, pickup_location, dropoff_location, selected_vehicle, selected_price,
                 license_plate, driver_name, vehicle_name):
        super().__init__(parent)
        self.parent = parent
        self.configure(bg="white")

        self.pickup_location = pickup_location
        self.dropoff_location = dropoff_location
        self.selected_vehicle = selected_vehicle
        self.selected_price = selected_price
        self.license_plate = license_plate
        self.driver_name = driver_name
        self.vehicle_name = vehicle_name

        self.backend = RideBackend("AIzaSyAOKrot@g067ji8DpUmxN3FdXRBfMsCvRQ")
```

- **Booked Ride Page**

Displays the details of the assigned driver and vehicle. Also provides the option to cancel the booking.

```
import sys
import os
import tkinter.font as Font

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from tkinter import *
from PIL import Image, ImageTk
from backend.ride_booking import RideBackend
from urllib.request import urlopen
from io import BytesIO

class BookedPage(Frame):
    """
    Displays the booked ride page for Move Out.
    Shows ride details, map, booking ID, estimated duration, and distance.
    """
    def __init__(self, parent, pickup_location, dropoff_location, selected_vehicle, selected_price,
                 booking_id, estimated_duration, license_plate, driver_name, vehicle_name):
        super().__init__(parent)
        self.parent = parent
        self.configure(bg="white")

        self.pickup_location = pickup_location
        self.dropoff_location = dropoff_location
        self.selected_vehicle = selected_vehicle
        self.selected_price = selected_price
        self.license_plate = license_plate
        self.driver_name = driver_name
        self.vehicle_name = vehicle_name

        self.backend = RideBackend("AIzaSyA0Krot@g067ji8DpUmxN3FdXRBTfMsCvRQ")
```

- **Ride Arrival Page**

Displays the "Rider on the way" status once a driver has been assigned.

```
import sys
import os
import tkinter.font as Font

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from tkinter import *
from PIL import Image, ImageTk
from backend.ride_booking import RideBackend
from urllib.request import urlopen
from io import BytesIO

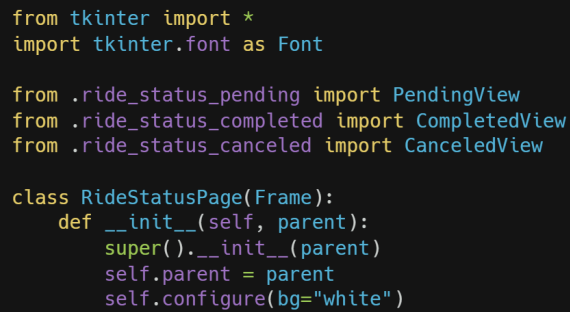
class RideArrivalPage(Frame):
    """
    Displays the Ride Arrival page for Move Out.
    Shows ride details, map, booking ID, estimated duration, and allows cancellation.
    """
    def __init__(self, parent, pickup_location, dropoff_location, selected_vehicle, selected_price,
                 license_plate, driver_name, vehicle_name):
        super().__init__(parent)
        self.parent = parent
        self.configure(bg="white")

        self.pickup_location = pickup_location
        self.dropoff_location = dropoff_location
        self.selected_vehicle = selected_vehicle
        self.selected_price = selected_price
        self.license_plate = license_plate
        self.driver_name = driver_name
        self.vehicle_name = vehicle_name

        self.backend = RideBackend("AIzaSyA0Krot@g067ji8DpUmxN3FdXRBTfMsCvRQ")
```

- **Ride Status Page**

Displays the ride history, organized into three sections:

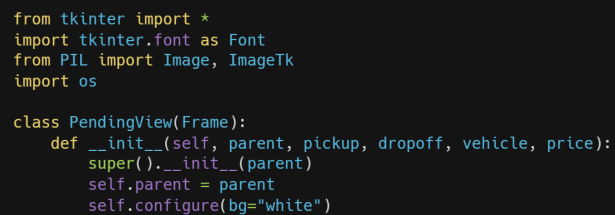


```
from tkinter import *
import tkinter.font as Font

from .ride_status_pending import PendingView
from .ride_status_completed import CompletedView
from .ride_status_canceled import CanceledView

class RideStatusPage(Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.parent = parent
        self.configure(bg="white")
```

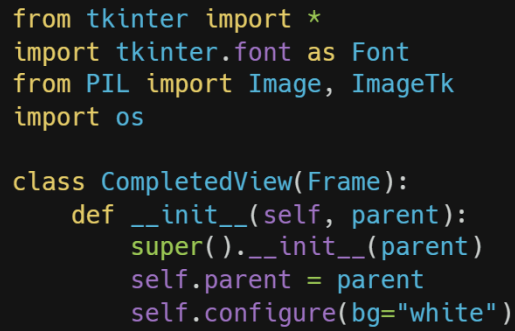
- **Pending Tab** – For the currently active or in-progress ride.



```
from tkinter import *
import tkinter.font as Font
from PIL import Image, ImageTk
import os

class PendingView(Frame):
    def __init__(self, parent, pickup, dropoff, vehicle, price):
        super().__init__(parent)
        self.parent = parent
        self.configure(bg="white")
```

- **Completed Tab** – For rides that have been marked as completed after the ride duration lapses.

A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in a light blue/cyan monospaced font. It shows imports for tkinter, PIL, and os, followed by a class definition for CompletedView that inherits from Frame. The __init__ method calls super().__init__(parent), sets self.parent to parent, and sets the background color to white.

```
from tkinter import *
import tkinter.font as Font
from PIL import Image, ImageTk
import os

class CompletedView(Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.parent = parent
        self.configure(bg="white")
```

- **Canceled Tab** – For rides canceled before completion.

A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is written in a light blue/cyan monospaced font. It shows imports for tkinter, PIL, and os, followed by a class definition for CanceledView that inherits from Frame. The __init__ method calls super().__init__(parent), sets self.parent to parent, and sets the background color to white.

```
from tkinter import *
import tkinter.font as Font
from PIL import Image, ImageTk
import os

class CanceledView(Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.parent = parent
        self.configure(bg="white")
```

- **Account Page**

Displays user profile details such as full name, email, and contact number.

```
from tkinter import *
import tkinter.font as Font

from backend.edit_account import EditAccountHandler
from backend.validators import InputValidator # Added for phone formatting

class AccountPage(Frame):
    def __init__(self, parent, current_user_phone):
        super().__init__(parent)
        self.parent = parent
        self.current_user_phone = current_user_phone # Store phone of logged-in user
        self.configure(bg="white")

        self.account_handler = EditAccountHandler() # Handler for backend operations
```

- **Interactive Map Page**

Displays static or simulated maps and routes, providing visual feedback for the user's ride.

```
def generate_static_map_url(self, pickup, dropoff, use_polyline=False):
    pickup_coords = self.get_coordinates(pickup)
    dropoff_coords = self.get_coordinates(dropoff)

    if not pickup_coords or not dropoff_coords:
        return None

    base_url = "https://maps.googleapis.com/maps/api/staticmap?"

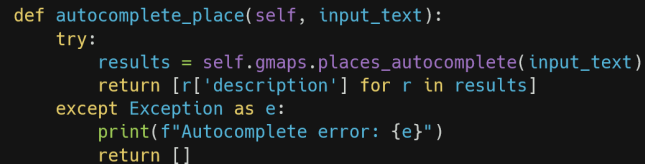
    markers = [
        f"color:green|label:P|{pickup_coords[0]},{pickup_coords[1]}",
        f"color:red|label:D|{dropoff_coords[0]},{dropoff_coords[1]}"
    ]

    return f"{base_url}{size_param}&{maptype_param}&{marker_params}&{path_param}&{key_param}"
```

Notable Features

- **Location Autocomplete and Suggestions**

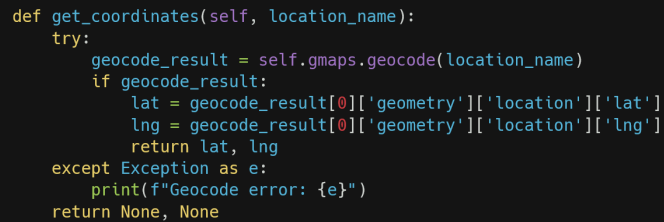
Users receive real-time, intelligent location suggestions through the Google Places API, improving accuracy and usability during pickup and drop-off selections



```
def autocomplete_place(self, input_text):
    try:
        results = self.gmaps.places_autocomplete(input_text)
        return [r['description'] for r in results]
    except Exception as e:
        print(f"Autocomplete error: {e}")
        return []
```

- **Accurate Location Pinning**

Inputs are converted to geographic coordinates to mark exact pickup and drop-off points using Google Geocoding.



```
def get_coordinates(self, location_name):
    try:
        geocode_result = self.gmaps.geocode(location_name)
        if geocode_result:
            lat = geocode_result[0]['geometry']['location']['lat']
            lng = geocode_result[0]['geometry']['location']['lng']
            return lat, lng
    except Exception as e:
        print(f"Geocode error: {e}")
        return None, None
```


- **Distance & Duration Calculation**

The app calculates actual driving distance (in kilometers) and estimated travel time using the Google Distance Matrix API.

```
def get_distance_km(self, origin, destination):
    try:
        result = self.gmaps.distance_matrix(origins=origin, destinations=destination,
        mode='driving')
        distance_text = result['rows'][0]['elements'][0]['distance']['text']
        distance_km = float(distance_text.replace(' km', '').replace(',', ''))
        return distance_km
    except Exception as e:
        print(f"Distance error: {e}")
        return 0.0

def get_estimated_duration(self, origin, destination):
    try:
        result = self.gmaps.distance_matrix(origins=origin, destinations=destination,
        mode='driving')
        duration_text = result['rows'][0]['elements'][0]['duration']['text']
        return duration_text
    except Exception as e:
        print(f"Duration error: {e}")
        return "--"
```

- **Fare Calculation Modeled After Real Ride-Hailing Apps**

Fare is computed using a formula similar to real-world pricing

```
def calculate_fare(self, distance_km, vehicle_type):
    # Base prices
    base_fares = {
        "Motorcycle": 50,
        "4-seater": 120,
        "6-seater": 150
    }

    base_fare = base_fares.get(vehicle_type, 50) # Default to 50 if unknown

    if distance_km <= 2:
        return base_fare
    elif distance_km <= 7:
        return base_fare + (distance_km - 2) * 10
    else:
        return base_fare + (5 * 10) + (distance_km - 7) * 15
```

- **Ride Status Handling Based on User Actions**

- **Pending** – When a ride is confirmed, it is immediately classified as pending

```
def on_confirm_clicked(self, event):
    if hasattr(self, 'selected_vehicle'):
        vehicle = self.selected_vehicle
        price = self.price_value.get("text")
        driver_info = self.backend.get_driver_info(vehicle)

        if hasattr(self.parent, "ride_status_page") and hasattr(self.parent.ride_status_page,
            "set_ride_details"):
            duration = self.backend.get_estimated_duration(self.pickup_location, self.dropoff_location)
            self.parent.ride_status_page.set_ride_details(
                self.pickup_location,
                self.dropoff_location,
                vehicle,
                price,
                duration
            )

            if hasattr(self.parent, "show_looking_page") and driver_info:
                license_plate = driver_info.get("license_plate", "")
                driver_name = driver_info.get("driver_name", "")
                vehicle_name = driver_info.get("vehicle_name", "")
                self.parent.show_looking_page(self.pickup_location, self.dropoff_location, vehicle,
                    price, license_plate, driver_name, vehicle_name)
            else:
                self.show_vehicle_warning()
```

- **Completed** – The ride automatically moves to the completed tab after the estimated ride duration has passed.

```
def complete_booking(db: DatabaseManager, booking_id: str):
    """Mark a ride as completed by moving it from pending to completed bookings."""
    result = db.execute_query(
        "SELECT * FROM pending_bookings WHERE booking_id = ?", (booking_id,)
    )

    if not result:
        print(f"❌ Booking {booking_id} not found.")
        return

    booking = result[0]

    db.execute_update(
        "INSERT INTO completed_bookings VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",
        booking
    )

    db.execute_update(
        "DELETE FROM pending_bookings WHERE booking_id = ?",
        (booking_id,)
    )

    print(f"✅ Booking {booking_id} marked as completed.")
    print(f"✅ Booking {booking_id} has been cancelled.")
```

- **Canceled** – If the user cancels before completion, the ride moves to the canceled tab while retaining all details.

```
def cancel_booking(db: DatabaseManager, booking_id: str):
    """Cancel a ride by moving it from pending to cancelled bookings."""
    result = db.execute_query(
        "SELECT * FROM pending_bookings WHERE booking_id = ?", (booking_id,)
    )

    if not result:
        print(f"✗ Booking {booking_id} not found.")
        return

    booking = result[0]

    db.execute_update(
        "INSERT INTO cancelled_bookings VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",
        booking
    )

    db.execute_update(
        "DELETE FROM pending_bookings WHERE booking_id = ?",
        (booking_id,)
    )

    print(f"✓ Booking {booking_id} has been cancelled.")
```

- **Back Navigation** – Clicking “Back” after reaching the Booked screen does not cancel or delete the ride; it remains visible in the Pending tab.

```
def on_back_clicked(self, event):
    if hasattr(self.parent, "ride_status_page"):
        self.parent.ride_status_page.restore_ride_info(
            self.pickup_location,
            self.dropoff_location,
            self.selected_vehicle,
            self.selected_price,
            self.duration_label.cget("text")
        )
    if hasattr(self.parent, "show_home_page"):
        self.parent.show_home_page()
```

- **Help Center Page**

Provides FAQs or user support content to help users navigate the system.



Testing and Evaluation

1. Testing Methods

- **Manual Testing:**
 - All testing was done manually by running the Python program using VSCode. The developers used terminal to observe outputs and check for any errors
- **Feature Testing:**
 - Each feature (like signup, login, ride booking, and account edit) was tested one by one to confirm it works properly
- **User Simulation:**
 - The developers acted as users and tried common scenarios like entering valid/invalid data to see if the app responded correctly
- **Regression Testing:**
 - After making changes or fixing bugs, we tested the app again to make sure everything still worked correctly

2. Bug Tracking

- **Tools Used:**
 - **VSCode terminal** to detect bugs and trace error messages
 - Notes to keep track of bugs manually
- **Bug Reporting Process:**
 1. **Bug Description:** Write down what went wrong

2. **Test Steps:** Recreate the error by repeating the action that caused it
3. **Severity Level:**
 - a. **Critical:** App crashes or feature completely fails
 - b. **Major:** Important feature doesn't work right
 - c. **Minor:** Small display or usability issue
4. **Status Tracking:**
 - a. **Open:** Bug found: not fixed yet
 - b. **Fixed:** Bug has been resolved
5. **Who Fixed It:** Noted by Marga Layo and Marjoy Caranto for accountability

3. Sample Test Cases and Results

Test Case ID	Test Description	Input	Expected Output	Actual Output	Status	Remarks
TC-001	Signup with valid credentials	First Name: Zach Last Name: Goodman Phone: 9082714653 PIN: 5720	The information is stored in user information database	The information is stored in user information database	Passed	None
TC-002	Signup with blank names	First Name: Last Name: Phone: 9998319955 PIN: 0525	The information will not be stored in user information database	The information is stored in user information database	Failed	Input validation for first and last name is missing; blank names were accepted.
TC-003	Signup with invalid phone number and PIN	First Name: Zyrele Last Name: Legaspi Phone: 99983199551 PIN: 052545	The information will not be stored in user information database	The information is not stored in user information database	Passed	None
TC-004	Signup with invalid phone number (less than 10 digits)	First Name: Zyrele Last Name: Legaspi Phone: 999831995 PIN: 0525	The information will not be stored in user information database	The information is stored in user information database	Failed	Phone number length validation is missing; numbers with fewer than 10

						digits were accepted and saved to the database.
TC-005	Signup with invalid phone number (less than 10 digits)	First Name: Zyrele Last Name: Legaspi Phone: 999831995 PIN: 0525	The information will not be stored in user information database	The information is not stored in user information database	Passed	None
TC-006	Signup with an existing phone number in the user information database	First Name: Zach Last Name: Goodman Phone: 9082714653 PIN: 5720	The information will not be stored in user information database	The information is not stored in user information database	Passed	None
TC-007	Login with an existing phone number and correct PIN	Phone: 9082714653 PIN: 5720	User is successfully logged in and redirected to the homepage	User is successfully logged in and redirected to the homepage	Passed	None
TC-007	Login with an existing phone number and incorrect PIN	Phone: 9082714653 PIN: 0000	User receives an error message indicating that the Phone number and PIN do not match up	User receives an error message indicating that the Phone number and PIN do not match up	Passed	None
TC-008	Login with non-existing phone number	Phone: 9001653475 PIN: 0000	User receives an error message indicating that the Phone number does not exist	User receives an error message indicating that the Phone number	Passed	None

				does not exist		
TC-009	Check if the user information is correctly on the Account Page after login	No input provided; verify that the current user data is displayed	The Account Page correctly displays the current user's information as stored in the database	The Account Page did not display the current user's information as stored in the database	Failed	The Account Page failed to display user information due to the missing data retrieval function
TC-010	Check if the user information is correctly on the Account Page after login	No input provided; verify that the current user data is displayed	The Account Page correctly displays the current user's information as stored in the database	The Account Page correctly displays the current user's information as stored in the database	Passed	None
TC-011	Edit the user's information	First Name: Zach → Sage Last Name: Goodman → Wilcox Phone: 9082714653 PIN: 5720	The user information is successfully updated and stored in the database.	The user information is successfully updated and stored in the database.	Passed	None
TC-012	Replace HTML+JS map with static map	Pickup: San Roque, Drop-off: Masinag	Static map image shown in same window	Static map shown, embedded within Tkinter	Passed	Solved by switching to Static Maps API; interactive map dropped due to JS restrictions
TC-013	Homepage front-end implementation (except navbar)	—	Homepage fields for pickup/drop-off with placeholders	Placeholders visible on Windows, not visible	Partial	macOS rendering issue; likely works fine on Windows

				on macOS		
TC-014	Google Maps API key not working	Any input	Location suggestions appear	No suggestions shown initially	Failed	API key missing; resolved after adding valid API key
TC-015	Enable location suggestions in pickup/dropoff	Partial input like "Antipolo"	Location suggestions appear	Suggestions appear correctly	Passed	Solved by enabling Places API via Google Cloud Console
TC-016	Price calculation per km	Pickup & drop-off 1km apart	Base fare + P15/km distance fee	Fare calculated but mislabeled	Passed	Labeled backend and frontend properly; logic works as expected
TC-017	Static map image route generation	Pickup: San Roque, Drop-off: Masinag	Route drawn on static map	One straight line shown	Partial	Initially missing detailed route; resolved later with correct polyline and zoom
TC-018	Directions API fails	Any pickup/dropoff	Directions route should load	Request denied	Failed	Legacy Directions API not enabled; fixed via Google Cloud Console
TC-019	Polyline route with zoom	Route via multiple coordinates	Map displays detailed route line with zoom	Works properly	Passed	Shows better route visualization
TC-020	pywebview to embed interactive map	map_view.html	Map appears within same application window	Opens in new window	Partial	pywebview launches separate window; doesn't

						embed in same frame
TC-021	tkhtmlview as a workaround for embedded maps	HTML with JS	Map embedded in same frame	No map shown	Failed	tkhtmlview doesn't support JavaScript; can't use it for embedding Google Maps
TC-022	Final polyline route with show_interactive_map.py	Updated HTML path and lat/lng route	Accurate interactive map loads	Works as expected	Passed	Fixed path and polyline route; map renders properly using pywebview
TC-023	Static Map as Final Map Display Method	Instead of using HTML and JavaScript	Static map image shown in app window	Map displays correctly using Google Maps Static API	Passed	Successfully generated map image through a direct Google Maps URL; avoids need for JS-based embedding

4. Evaluation Summary

The testing of the user account management features was largely successful. Valid signup and login scenarios worked correctly, with successful account creation and redirection. However, there were issues with input validation: blank first and last names, as well as phone numbers with fewer than 10 digits, were accepted. The Account Page displayed user information correctly in most cases but failed in TC-009 due to a missing data retrieval function, which was fixed in TC-010. Editing user information, including name and phone number, was successfully implemented. Overall, the system functions well, but improvements are needed in input validation and the data display mechanism for consistency and reliability.

The testing of TC-012 to TC-023 focused on map display, location services, routing, and fare computation. The replacement of the interactive map with a static image (TC-012, TC-023) was successful, resolving compatibility issues. Location suggestions and fare calculations worked as expected after resolving API configuration issues (TC-014 to TC-016). Routing accuracy improved from a basic line to a detailed polyline with zoom (TC-017, TC-019, TC-022). Some attempts to embed interactive maps (TC-020, TC-021) were unsuccessful due to JavaScript limitations. Overall, key location and mapping features function reliably using the static map approach.

Results and Discussion

The system's main features - signup, login, and user information editing functions well, but some issues in input validation were found including accepting blank names, and incomplete phone numbers and were flagged as bugs for improvement. Account display errors were also resolved. For map and location features, the app shifted from an interactive type to just a static one due to JavaScript limitations, which fixed incompatibility issues. Location suggestions, routing, and fare calculations have functioned correctly after doing some configurations in API. Some embedding attempts have failed, but the final map using a static image has been effective. Overall, the system is functional but requires better data input validation and improved support for embedded maps.

Behind a successful program is a heavy load of challenges and obstacles during the development phase. These include having conflicts with putting the google map in the booking page, issues with showing location suggestions when user inputs a pick-up and drop-off location, incorrect logic and labeling on the map, issues with the pricing because of not properly labeled variables in the front and backend, incorrect pricing index when the pick-up and drop-off location is between countries, placing an interactive Google map, difficulty in aligning the text and buttons after the animations due to different monitor scale and resolution, problems with making all the functions fit to one window only and not in separate ones, as well as the original planned font styles and weight couldn't be applied. The indicated situations are just a few of the many difficulties the team experienced while the program was being developed, but through creative thinking and resourcefulness, the team was able to make solutions by making use of several third-party extensions and applications and by fixing certain conditions in the computer settings.

Conclusion

Throughout the development of *Move Out*, the team gained valuable experience in applying core programming concepts to a real-world system. The developers strengthened their understanding of object-oriented programming by using classes, inheritance, encapsulation, and polymorphism to organize and structure the system. Working with Python and Tkinter gave the developers an insight into designing

interactive graphical user interfaces, while integrating SQLite taught the developers how to manage persistent data effectively. The use of Google Maps APIs challenged the developers to handle external services, manage API responses, and work with geolocation data—skills that are essential in modern software development.

The project also highlighted the importance of planning and coordination. From brainstorming the app's design and assigning roles, to resolving unexpected bugs and interface issues, every phase required strong communication and problem-solving as a team. The developers encountered multiple obstacles such as incorrect data mapping, display inconsistencies due to screen resolution differences, and issues with validating user input, but we were able to overcome them through collaboration, research, and systematic debugging using the VSCode terminal. These challenges pushed the developers to think critically and reinforced their ability to work under pressure while maintaining code quality and user experience.

Future Work

To enhance the overall functionality, scalability, and user experience of the system, several additional features and improvements are recommended for future prototype versions.

The development of a driver-side page and interface that allows real-time interaction between drivers and passengers could include functions such as live location tracking and acceptance of booking. Also, addressing the current issue where bookings can extend beyond national borders or cross into bodies of water. This can be resolved by limiting booking distances and applying geographic boundaries to ensure accurate fare computations.

To enhance user account security, it is recommended to integrate advanced authentication features such as email verification, password recovery, and two-factor authentication. Adding digital payment gateways would also improve convenience and security of payment by enabling fare payments through platforms like GCash, PayMaya, or credit/debit cards. Expanding SIM card compatibility and support for international phone numbers would allow a broader range of users to register.

Additional features such as offline booking functionality which allows requests to be stored and synced when internet connectivity is restored would improve system reliability. Implementing push notifications or SMS alerts would enhance communication by sending ride confirmations and updates.

Moreover, the system can be improved by utilizing tools that enhance interface transitions and overall user experience. Incorporating animations, smoother page navigation, and responsive design elements can create a more intuitive and engaging interaction for users, as well as adding in desktop-based systems and environments.

Lastly, a user rating and feedback system could also help maintain service quality and accountability.

These proposed improvements aim to align the system with modern ride-booking standards while keeping its focus on accessibility, practicality, and student-centered design.

References

(2022). Capture embedded google map image with Python without using a browser

[Review of Capture embedded google map image with Python without using a browser]. Stack Overflow.

<https://stackoverflow.com/questions/7490491/capture-embedded-google-map-image-with-python-without-using-a-browser>

Tripathi, S. (2024). *How to use the Google Maps API in Python: a quick guide*.

Apify Blog. <https://blog.apify.com/google-maps-api-python/>

Appendices

Appendix A: Installation/User Manual

• System Requirements

- Operating System: Compatible with Windows, macOS, or Linux
- Required Packages / Software:
 - ❖ Python (version 3.10 or above recommended)
 - ❖ SQLite
 - ❖ tkinter (usually comes bundled with Python, but may require manual installation on some systems)

• Installation Steps

1. Clone the repository:
 - Open Git Bash (or any terminal) and run the following command:
`git clone https://github.com/mjcarant0/move_out.git`
 - Choose and remember the directory where you cloned the repository.
2. Navigate to the project folder:
 - `cd move_out`
3. Open the project folder:
 - Use Visual Studio Code or any IDE of your choice to open the folder.
 - Locate the main runnable file named `move_out.py`.
4. Run the application:
 - In your terminal or IDE, execute the main script:

```
python move_out.py
```

- Make sure you're in the project folder when running this command.

Appendix B: Description of the App Name

App Name: Move Out

Description:

The name Move Out was chosen as a playful twist on words, inspired by the well-known Philippine moto-taxi app Move It. The team wanted something catchy and culturally familiar. They aimed for a name that would reflect the app's core purpose of helping users book rides and manage transport while on the go. At the same time, they wanted it to carry a touch of humor. By flipping Move It into Move Out, the name immediately suggests movement or leaving a place, which perfectly matches the app's goal of bringing users from one location to another.

However, Move Out is more than just a clever name. At its core, the project demonstrates object-oriented programming (OOP) concepts in action. Built using Python, the system applies key principles such as classes, inheritance, encapsulation, and polymorphism in a practical, real-world setting. The app features a user-friendly graphical interface built with Tkinter. It features a user-friendly graphical interface built with Tkinter, a reliable SQLite backend for data storage, and API integration with various Google Maps services for route generation and distance calculation.

By combining humor, cultural relevance, and strong programming foundations, Move Out becomes more than a ride-booking app. It is a creative and educational project that reflects both the technical skills and the personality of its developers. It stands as a parody with a purpose, built on solid software engineering principles.

Appendix C: Logo Description



Description:

The Move Out logo is a playful and thoughtful take on the branding of the well-known Philippine moto-taxi app Move It. While we took inspiration from its familiar helmet icon, we gave it a fresh identity by changing the perspective to a front-facing view. This change may seem subtle, but it carries a powerful message.

Unlike the original side-view design, the front-facing helmet puts the user directly in focus. It creates a sense of connection, movement, and control. The user is no longer just watching from the sidelines but is now fully in charge of their journey. This perspective reflects the belief that technology should empower people and place them at the center of the experience.

The developers also chose a pink color palette, not just for visual appeal, but to reflect the identity of the team behind the project. Created by an almost all-girl development team, the soft tones and warm aesthetic express their creativity, uniqueness, and confidence. In a tech space that is often male-dominated, this choice challenges expectations while embracing who we are. Beneath the friendly look is a solid system built with clean Python code and strong object-oriented programming practices.

To bring the purpose of the app into the design, we added a small map pin. It highlights the theme of travel and navigation, reminding users that Move Out is here to help them get from one place to another with clarity and ease.

Overall, the Move Out logo is more than just an image. It represents direction, identity, and the spirit of the team behind it. It is a simple yet meaningful reflection of a product designed with care, confidence, and a user-first mindset.