Group Project Final Report

# Project Title: Ride Booking System

# Bachelor of Science in Computer Engineering:

Object Oriented Programming

# 1st / 1-7 / \*Name Pending\* / Group 6:

# Team Members:

| Name | Student ID | Role |
| --- | --- | --- |
| Ron Andre Malabuyoc | 2024-01954-MN-0 | Leader, Backend Developer, and UI Designer |
| Maria Teresa Mae De Jesus | 2024-04831-MN-0 | UI/UX Designer and Developer |
| JD Angelo Soon | 2024-04621-MN-0 | UI/UX Designer and Developer |
| Lord Van Suva Baltazar | 2024-02848-MN-0 | Backend Developer |
| Kirsten Caryl Boncales | 2024-05011-MN-0 | Backend Developer |
| Matthewe Gabriel | 2024-01055-MN-0 | Database Developer and Bug Tester |
| Ezekiel Zio Gozon | 2024-04217-MN-0 | Backend Developer |

# Instructor/Adviser: Avena Godofredo

# Date Submitted:

July 4, 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

Our Ride Booking Program was made with the thought of being an indie competitor to industry staples like Grab, Angkas, and MoveIt. Our systems are meant to be an all-in-one hub for Ride Sharing to reduce the need of moving between apps and maximize convenience for the end user.

# Objectives

#### General Objective

* To develop an easy to understand, functional, and fun ride-booking application that connects passengers to drivers.

#### Specific Objectives

1. **User Registration and Profile Management:**
   * To allow riders and drivers to create and manage personal profiles, including contact information, vehicle details (for drivers), and payment preferences.
2. **Location-Based Ride Booking:**
   * To implement a geolocation feature that enables riders to find nearby drivers based on their request.
3. **Vehicle Type Selection:**
   * To provide users with the option to choose from multiple vehicle types (car, motorcycle, van) based on their preferences and available options, offering flexibility to riders.
4. **Payment Integration:**
   * To integrate secure payment methods such as GCash, PayPal, and COD, enabling riders to complete their payments seamlessly after the ride, while allowing drivers to track their earnings.
5. **Real-Time Ride Tracking:**
   * To incorporate a real-time tracking feature, allowing riders to monitor the location of their vehicle and drivers to navigate to the pickup location, improving the overall ride experience.
6. **User Interface (UI) and User Experience (UX):**
   * To design a clean and intuitive interface for both the rider and driver sides, ensuring ease of navigation and a smooth user experience.
7. **Testing and Quality Assurance:**
   * To rigorously test the system for bugs, performance issues, and usability problems, ensuring that the application meets the expected quality standards before deployment.

# Scope and Limitations

**Scope:**

**User Registration and Profile Creation**

1. Riders and drivers will be able to create and manage their profiles, including personal information and payment details.

**Request-Based Services**

1. Passengers will be able to receive their ride once they book a ride and there is a driver ready and available.

**Ride Booking and Vehicle Selection**

1. Riders will be able to book a ride by selecting the vehicle type (car, motorcycle, van), and the system will assign a nearby driver.

**Payment Integration**

1. Riders will be able to pay for their rides using integrated payment options such as GCash, PayPal, and credit/debit cards. Cash on Delivery (COD) will also be supported.

**Ride Tracking**

1. Riders can track their vehicle’s location in real-time during the ride. Drivers can navigate to the rider’s location using GPS integration.

**Ratings and Reviews**

1. After each ride, both riders and drivers will be able to rate each other on a 5-star scale and leave feedback.

**Admin Dashboard**

1. Admins will be able to manage users (drivers and riders), monitor ride statistics, and manage payments.

**Limitation:**

**Platform Limitation**

1. The application is currently designed for desktop use only, with no mobile version available.

**Limited Payment Methods**

1. The system supports only a few payment methods (GCash, PayPal, and COD). Additional payment gateways (e.g., credit/debit card integrations, Google Pay, etc.) were not included due to time constraints.

**Lack of Advanced AI Features**

1. The application does not incorporate advanced AI-driven features such as ride price prediction, driver-rider matching algorithms, or real-time traffic data integration. These could be considered for future enhancements.

**Geolocation Accuracy**

1. The distance location from our Geolocation and Map integration gives a straight route, not taking into account roads, traffic, and possible blockages

**Limited Testing Coverage**

1. The system has been tested with a limited number of test cases, focusing primarily on functional testing for core features like ride booking and payment. User acceptance testing was not extensively conducted due to time constraints.

**Security Constraints**

1. There is no robust security system besides giving a distinct ID for each booked ride. Payment security and account security are not implemented in the program

**No Multilingual Support**

1. The app is only developed in English and offers no other language options

**Limited User Load**

1. The system has been designed for a small to medium user base. Scalability to accommodate a large number of simultaneous users may be limited without further optimization.

**Robust GPS System**

1. The app does not have an implemented GPS system that uses its relative location to find where the passenger is, as users have to manually input their pickup location.

# Methodology

The development of the Ride Booking Application followed a rushed development, though planning was conceived early, the implementation of such plans was delayed until recently because of exams.

#### 1. Planning Phase

During the planning phase, we decided to divide the work into parts that each member is capable of and experienced but still left room for members to approach any aspect of the program if they so wished or needed to. The main concepts and ideas of the program were brainstormed by the entire group, adding features that would be great to add to the program. In this phase, we also committed to using Github Packages like Custom Tkinter (For GUI) and Tkinter MapView (For Google Maps integration) for an easier and more pleasing look to our program. We also committed to utilizing GitHub as our main platform to share our code and track progress.

#### 2. Design Phase

The design phase was divided into two key tasks:

* **User Interface (UI) Design:** One team member, specializing in design, is responsible for creating a mockup of the program with no functionality but gives the idea of what the program could look and flow together.
* **Database Design:** Another team member with experience in database management focused on designing the database system. The database utilized Python and saved those profiles into CSV files.

#### 3. Development Phase

The development phase was divided into several key tasks, each assigned to specific group members based on their skills and strengths:

* **Backend Development:** The set of members set to develop each aspect of the backend, such as the vehicle management, ride booking, and file handling, worked closely to integrate well together
* **Frontend Development:** The members set on creating and building the GUI are set to translate and implement the code from the backend and apply it seamlessly to buttons and other interactive functions.
* **Geolocation Integration:** Members researching how to integrate our TkinterMapview package and Geopy into the program so we could calculate distance and the fare for booked rides.
* **Quality Assurance (QA) and Testing:** Every member had access to the GitHub repository, so everyone was able to check and debug any possible deficiencies in the program. These deficiencies are communicated through and given a solution as soon as they are found.

#### 4. Implementation and Deployment

During the implementation phase, the group collaborated closely to merge the individual components and ensure they worked together effectively. We utilized **GitHub** for version control, ensuring that all team members could collaborate without overwriting each other's work. Code reviews were conducted regularly to maintain high-quality standards.

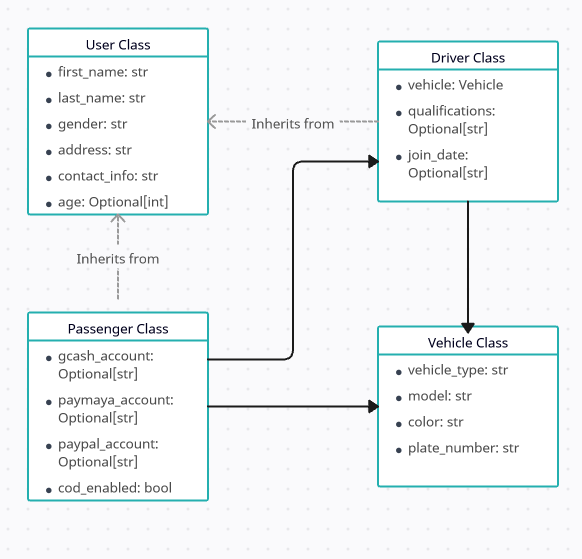
Once the core features were integrated, the team deployed the application for internal testing. The group worked together to fix any bugs and make adjustments based on feedback from the testers.

#### 5. Final Review and Documentation

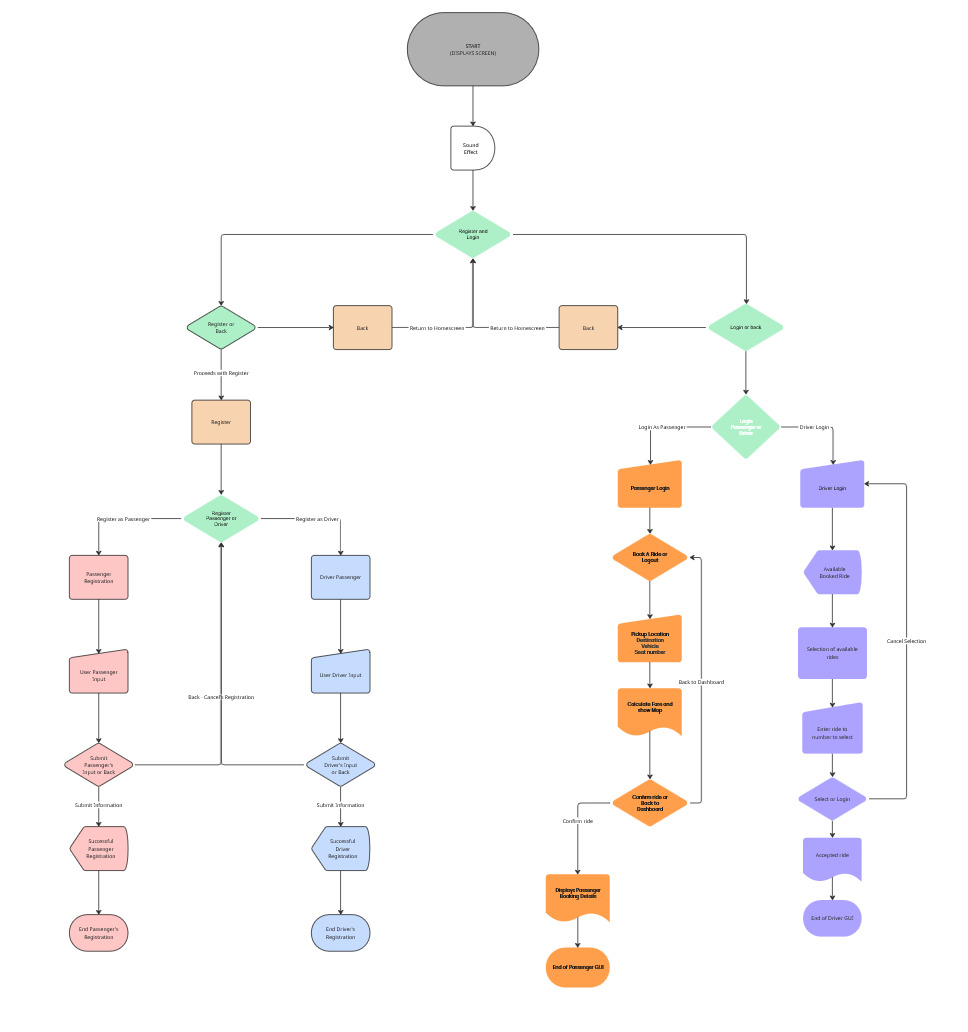
At the final stage, the team came together to review the completed application, ensure all features were functioning as expected, and prepare project documentation. Each member contributed to writing the user manual, system design documentation, and technical reports, which detailed the architecture, database, and codebase.

# System Design

Class Diagram:



Flow Chart:



# Technologies Used

Duck Dash was developed using Python 3.12.6 as the core language, with Visual Studio Code and GitHub serving as our primary development and collaboration tools. For the user interface and functionality, we used several third-party libraries alongside modules from the Python Standard Library. These technologies allow the app’s GUI, map visualization, geolocation, sound playback, and data storage capabilities.

Libraries used that are not in the Standard Python Library:

* geopy
* tkintermapview
* customtkinter
* PIL
* pygame

geopy and tkintermapview are libraries that are used in tandem to compute the distance between the pickup and drop-off location with geopy and visualize it using the tkintermapview

customtkinter allows us to use it in combination with the standard tkinter with new, modern, and fully customizable widgets

PIL library allows us to use different types of image file formats to be used in our program, which is mainly our logo at the opening sequence

pygame library is used to put our opening jingle in the opening sequence

Libraries used that are in the Standard Python Library:

* tkinter
* datetime
* csv
* os

The tkinter is the main foundation of making the user interface of Duck Dash as part of our curriculum in learning Python

The datetime library is used to assign the time of registration for drivers as a part of their parameter for their Driver class

csv library is the main file handling tool that we use to save and load the user, drivers, and passengers' information that we use for our program

os library lets us path the Duck Dash application to find the necessary files in the same folder that it is in

# Implementation

Provide details on the modules and features developed. Screenshots and code snippets can be included.

class User

* Used to collect parameters for registering either a passenger or a driver:
  + First Name
  + Last Name
  + Gender
  + Address
  + Contact Information
  + Age

Code snippet for class User

class User:

def \_\_init\_\_(self, first\_name, last\_name, gender, address, contact\_info, age=None):

self.first\_name = first\_name

self.last\_name = last\_name

self.gender = gender

self.address = address

self.contact\_info = contact\_info

self.age = age

class Passenger(User)

* If the user registered as a passenger, it inherits the class User parameters and adds the information about their method of payment
  + Gcash
  + Paymaya
  + Paypal
  + Cash on Delivery

Code snippet for class Passenger (User)

class Passenger(User):

def \_\_init\_\_(self, first\_name, last\_name, gender, address, contact\_info,gcash\_account=None, paymaya\_account=None, paypal\_account=None, cod\_enabled=True):

super().\_\_init\_\_(first\_name, last\_name, gender, address, contact\_info)

self.gcash\_account = gcash\_account

self.paymaya\_account = paymaya\_account

self.paypal\_account = paypal\_account

self.cod\_enabled = cod\_enabled

class Driver(Used)

* If the user registered as a driver, it inherits the class User parameters and adds the information as a driver, which is then saved in the drivers.csv file
  + Vehicle
  + Qualifications (Driver's License)
  + Join Date

Code snippet for class Driver(User)

class Driver(User):

def \_\_init\_\_(self, first\_name, last\_name, gender, address, contact\_info, vehicle, qualifications=None, join\_date=None):

super().\_\_init\_\_(first\_name, last\_name, gender, address, contact\_info)

self.vehicle = vehicle

self.qualifications = qualifications

self.join\_date = join\_date or datetime.now().isoformat()

class Vehicle

* Part of registration a driver and saved in the same drivers.csv file
  + Vehicle Type
  + Model (Vehicle model)
  + Color (Vehicle color)
  + Plate Number

Code snippet for class Vehicle

class Vehicle:

def \_\_init\_\_(self, vehicle\_type, model, color, plate\_number):

self.vehicle\_type = vehicle\_type

self.model = model

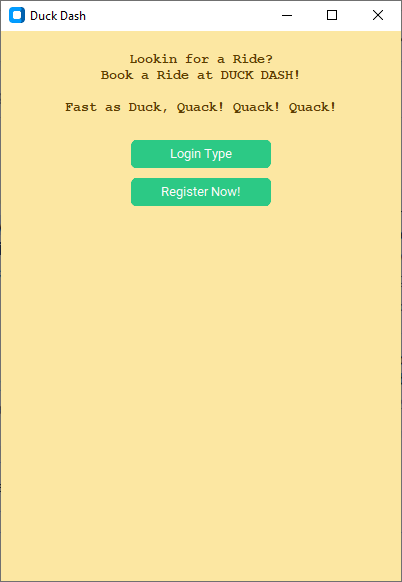
self.color = color

self.plate\_number = plate\_number

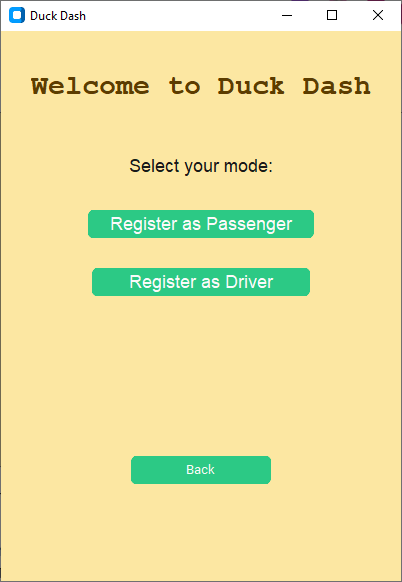
class DuckDashApp(ctk.CTk)

* Main class that will initiate the start of the user interface of Duck Dash
* It will run functions in succession based on what the user has clicked

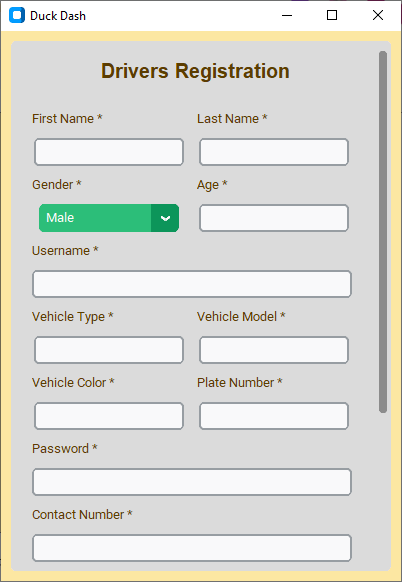
Starting Menu after the load-up animation from the class DuckDashApp instance



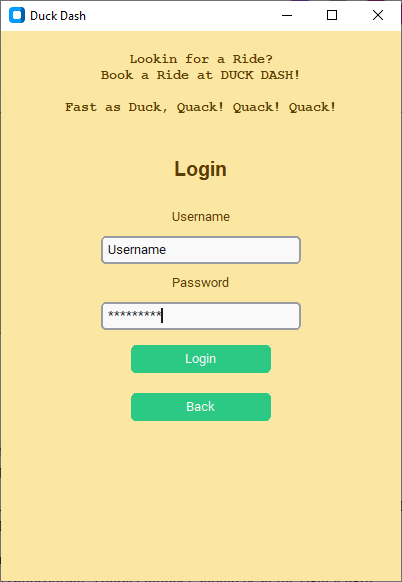
Click Register Now! Will prompt you to choose to register either as a driver or a passenger



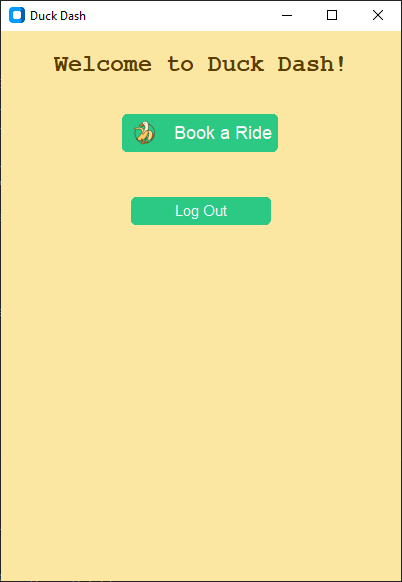
In this example we’ll register as a driver and it will ask as to fill up the forms for registration which will use the class Driver(User) to make an object and be put in the drivers.csv in the same folder

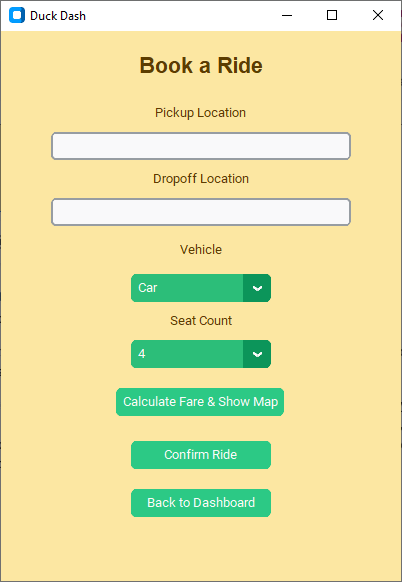


Going back from the start and now clicking Login Type and Login as Driver we will be prompted to enter the login credentials



After logging in with using the credentials that you’ve put in registering, you can now use it to book a ride





Picking the login as driver from the other side will prompt you with a different screen which is a dashboard for

Picture yung gawa ni jd sooner

# Testing and Evaluation

Describe testing methods, bug tracking, and provide sample test cases and results.

We have mostly done manual testing as the logic and the possible things that we can do with the program are very small and manageable.

We also tested common error occurring inputs such as not putting anything in the entry spaces, checking if the age is appropriate, contact number should be at the right length, logging in with the wrong password, and even using the login credentials from the drivers to the passengers and vice versa.

| Bugs | Description | Fix |
| --- | --- | --- |
| Saving data into .csv problems | Not properly saving into their respective csv files | Using from os import \* is different from import os since the open() function is being overriden |

# Results and Discussion

The final program meets most of the plans we set out to implement. It manages to create and read profiles for both Passenger and Driver users, with their own separate dashboards. The main purpose of the project, the ability to book a ride, receive its distance and fare and have someone accept and take in a passenger works well, through its simple but effective database system.

Though the product could work better with a robust GPS implementation, that feature is just beyond our means at the moment, the distance calculation could also use improvement as it calculates distance in a straight line and doesn’t take into account any street routes.

All in all the program works well, and is a small glimpse of the potential of what this program could be with further development.

# Conclusion

During the program, as a group we value our teamwork and cooperation all throughout, this project has brought us to learn about how everyone approaches each of their responsibilities and their strengths and weaknesses in programming. By the end we managed to end with a product that matches our wants while still finding fun in the process. Though for all of us, there will still be things to improve in many aspects like planning, implementing and cooperating, this project will be a learning experience and we’ll use it to create an even greater product that will please more.

# Future Work

Additional features to include

* Mobile Port (Android and iOS)
* A robust GPS system
* A stronger security system
* Distance and route calculations that take into account routes, roads blockages, and traffic
* Multi-lingual support

# References

Akabekobeko. (2023). *TkinterMapView documentation*. GitHub.<https://github.com/TomSchimansky/TkinterMapView>

GeeksforGeeks. (2024). *Object-oriented programming in Python*. GeeksforGeeks.<https://www.geeksforgeeks.org/python-object-oriented-programming/>

GeoPy Contributors. (2023). *GeoPy documentation*. GeoPy.<https://geopy.readthedocs.io/>

Pygame Community. (2023). *Pygame documentation*. Pygame.<https://www.pygame.org/docs/>

Python Imaging Library. (2023). *Pillow documentation*. Pillow.<https://pillow.readthedocs.io/>

Python Software Foundation. (2023a). *Python programming language*. Python.org.<https://www.python.org/>

Python Software Foundation. (2023b). *Tkinter documentation -- Python interface to Tcl/Tk*. Python.org.<https://docs.python.org/3/library/tkinter.html>

TomSchimansky. (2023). *CustomTkinter documentation*. GitHub.<https://github.com/TomSchimansky/CustomTkinter>

RonAndreMalabuyoc. (2025). *Ride\_booking\_system\_project.* GitHub. <https://github.com/RonAndreMalabuyoc/ride_booking_system_project.git>

# Appendices

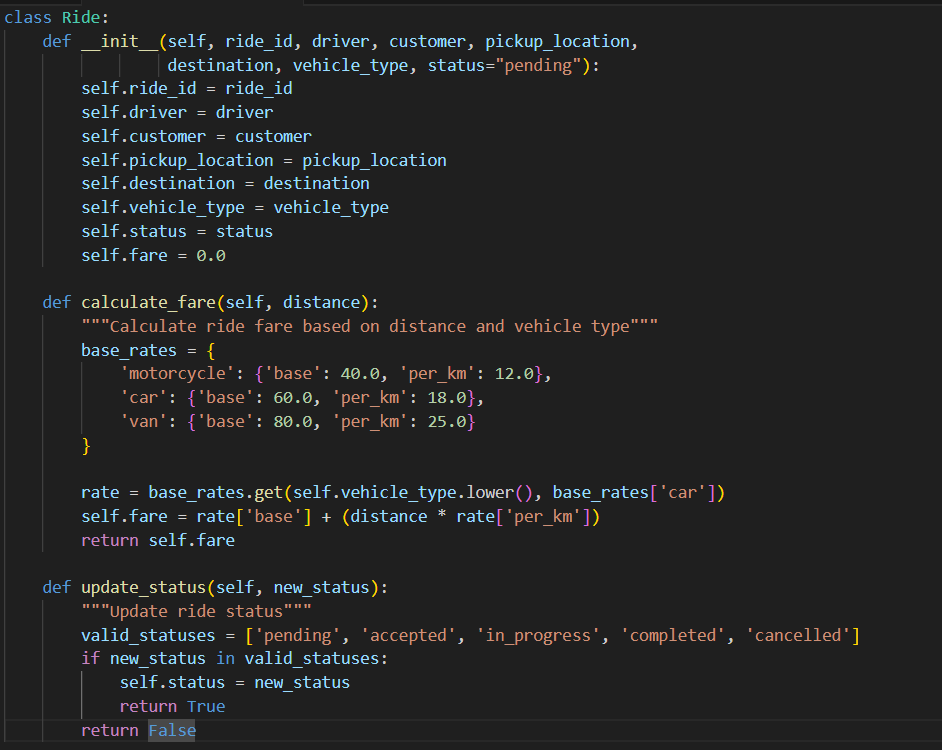
### Appendix A: System Architecture Diagram

The system architecture follows a hierarchical structure with clear separation of concerns:

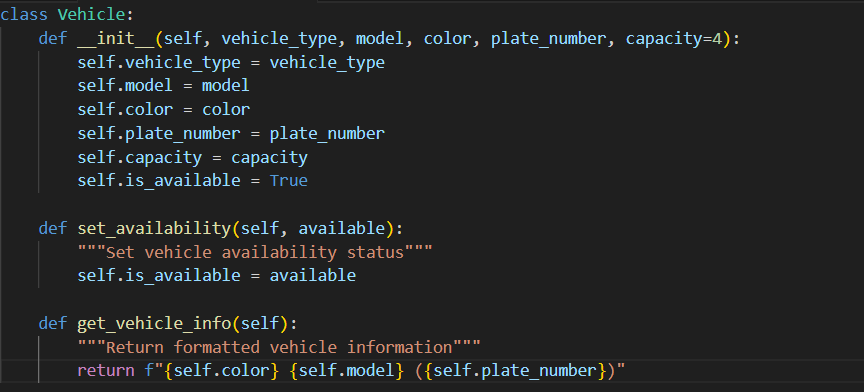
* **Presentation Layer**: CustomTkinter-based GUI components
* **Business Logic Layer**: Core application classes and ride management
* **Data Access Layer**: CSV-based file handling and data persistence
* **External Services Layer**: Geolocation and mapping services integration

### Appendix B: Complete Code Implementation

**Ride Management System**

****

**Vehicle Management System**



### Appendix C: User Interface Screenshots

**Main Application Interface**

* Launch screen with animated logo and background music
* Clean, modern design using CustomTkinter styling
* Navigation menu with passenger and driver options

**User Registration Forms**

* Separate registration forms for passengers and drivers
* Input validation and error handling
* Profile picture upload capability

**Ride Booking Interface**

* Interactive map for location selection
* Vehicle type selection with pricing information
* Payment method selection and confirmation

**Administrative Dashboard**

* User management interface
* Ride statistics and analytics
* Payment processing and dispute resolution

### Appendix D: Installation and Configuration Guide

**System Requirements**

* **Operating System**: Windows 10/11, macOS 10.15+, or Linux Ubuntu 18.04+
* **Python Version**: 3.12.6 or higher
* **RAM**: Minimum 4GB, recommended 8GB
* **Storage**: 100MB available space
* **Network**: Internet connection for geolocation services

**Installation Steps**

**Download and Install Python** bash  
# Verify Python installation

1. python --version

**Clone Repository** bash  
git clone https://github.com/group6/duck-dash.git

1. cd duck-dash
2. Install Dependencies  
    bash  
   pip install -r requirements.txt
3. Configuration Setup
   * Create config.py file with API keys
   * Configure database connection settings
   * Set up logging preferences
4. Run Application  
    bash  
   python duck\_dash.py

**Dependencies List**

customtkinter>=5.2.0

tkintermapview>=1.29

geopy>=2.3.0

Pillow>=10.0.0

pygame>=2.5.0

pandas>=1.5.0

numpy>=1.24.0

### Appendix E: User Manual

**Getting Started Guide**

**For New Users:**

1. Launch the Duck Dash application
2. Select "New User Registration"
3. Choose account type (Passenger or Driver)
4. Complete the registration form
5. Verify account information
6. Begin using the application

**Passenger Instructions:**

*Booking a Ride:*

1. Click "Book a Ride" from the main menu
2. Enter pickup location or use current location
3. Enter destination address
4. Select preferred vehicle type
5. Choose payment method
6. Review fare estimate
7. Confirm booking
8. Track driver location in real-time

*Managing Profile:*

1. Access "Profile Settings" from main menu
2. Update personal information
3. Add or modify payment methods
4. View ride history
5. Update emergency contacts

**Driver Instructions:**

*Getting Started:*

1. Complete driver registration with vehicle information
2. Upload required documents (license, vehicle registration)
3. Wait for account verification
4. Set availability status

*Accepting Rides:*

1. Enable "Available" status
2. Receive ride notifications
3. Review ride details (pickup, destination, fare)
4. Accept or decline ride requests
5. Navigate to pickup location
6. Confirm passenger pickup
7. Navigate to destination
8. Complete ride and process payment

*Vehicle Management:*

1. Access "Vehicle Settings"
2. Update vehicle information
3. Set vehicle availability
4. Track earnings and performance

**Administrative Functions:**

*User Management:*

1. Access admin dashboard
2. View user accounts and statistics
3. Handle user verification
4. Manage account suspensions

*System Monitoring:*

1. Monitor ride statistics
2. Track payment transactions
3. Generate reports
4. Handle customer support requests

### Appendix F: Technical Specifications

**Database Schema**

*Users Table (users.csv):*

user\_id, first\_name, last\_name, gender, address, contact\_info, age, user\_type, registration\_date

*Drivers Table (drivers.csv):*

driver\_id, user\_id, vehicle\_type, model, color, plate\_number, qualifications, join\_date, status

*Rides Table (rides.csv):*

ride\_id, driver\_id, passenger\_id, pickup\_location, destination, vehicle\_type, status, fare, booking\_time, completion\_time

*Payments Table (payments.csv):*

payment\_id, ride\_id, user\_id, payment\_method, amount, transaction\_status, payment\_date

**API Endpoints**

*User Management:*

* register\_user(user\_data): Register new user
* authenticate\_user(credentials): User login
* update\_profile(user\_id, updates): Update user information
* get\_user\_profile(user\_id): Retrieve user data

*Ride Management:*

* create\_ride(ride\_data): Create new ride booking
* update\_ride\_status(ride\_id, status): Update ride status
* get\_ride\_history(user\_id): Retrieve user ride history
* calculate\_fare(distance, vehicle\_type): Calculate ride cost

*Location Services:*

* get\_current\_location(): Get user GPS coordinates
* calculate\_distance(pickup, destination): Calculate route distance
* find\_nearby\_drivers(location, radius): Find available drivers

**Security Considerations**

*Data Protection:*

* User passwords hashed using SHA-256
* Personal information encrypted in storage
* Payment details handled through secure APIs
* Session management with timeout controls

*System Security:*

* Input validation and sanitization
* SQL injection prevention
* Cross-site scripting protection
* Rate limiting for API calls

**Performance Optimization**

*Caching Strategy:*

* User session data cached in memory
* Frequent database queries optimized
* Map tiles cached for offline access
* Driver location updates throttled

*Resource Management:*

* Memory usage monitoring
* Automatic garbage collection
* File handle management
* Network connection pooling