Table of Contents

[Introduction 1](#_Toc199424786)

[Project Background 2](#_Toc199424787)

[Scope 3](#_Toc199424788)

[Exclusions and Limitations 5](#_Toc199424789)

[Features 6](#_Toc199424790)

[Installation and Setup 9](#_Toc199424791)

[Project Files and Their Purpose 10](#_Toc199424792)

[Diagrams 11](#_Toc199424793)

[License 16](#_Toc199424794)

[Development 17](#_Toc199424795)

[Tools 19](#_Toc199424796)

[Future Maintenance and Support: 19](#_Toc199424797)

[Conclusion 21](#_Toc199424798)

[ReadMe 21](#_Toc199424799)

[References 23](#_Toc199424800)

# Introduction

The Car Rental System is Python-based software that enables fast and easy car rental operations for rental agencies. It addresses the challenges of managing and processing rental applications, handling customer registrations, and overseeing payments—activities that can be boring and prone to error when conducted manually. This system aims to boost efficiency, reduce mistakes, and enhance the customer experience.

It is built on a modular framework. The User Management module handles customer registrations, logins, and loyalty points tracking. The Car Management module allows administrators to add, view, modify, and remove vehicles. The Booking Manager streamlines the rental process—from selecting a car and validating the rental period to processing payments and updating user loyalty. Additionally, it employs a robust database layer through Firebase Firestore for scalable, cloud-based data storage, ensuring accurate recording of each rental request and transaction as individual records.

The Car Rental System combines front-end interactions with back-end solutions. This project focuses on security, reliability, and user-friendly interfaces, offering a comprehensive and practical application that meets the needs of both customers and rental service providers.

# Project Background

Traditional car rental operations, which often relied on manual booking, paperwork, and phone-based customer interactions, have become less efficient due to rising customer expectations for convenience, speed, and seamless service.

The Car Rental System project was considered to address the challenges faced by both rental companies and customers. For companies, managing vehicles involves complex tasks, including scheduling, tracking vehicle availability, processing payments, and maintaining accurate customer records. These tasks have historically been managed through disparate systems or even paper-based logbooks, resulting in errors, delays, and increased operational costs. For customers, the process of renting a car can often be clumsy, with confusing interfaces, inconsistent booking processes, and limited transparency in vehicle availability and pricing.

With these challenges in mind, the project aims to provide a comprehensive, integrated solution to automate key aspects of the car rental process. This project not only reflects the digital evolution of the car rental industry but also serves as a practical application of modern software engineering principles in solving real-world business challenges.

# Scope

The scope of the Car Rental System project outlines the functionalities and services the system will cover. This project is designed as both a practical solution for managing car rental operations and a demonstrative academic exercise in modern software engineering practices.

1. **User Management:**
   * **Registration & Authentication:** Users (both customers and administrators) will be able to create accounts, securely log in, and manage their profiles.
   * **Loyalty Points:** The system tracks and displays loyalty points for customers, including the functionality to redeem points for rental discounts.
2. **Car Management:**
   * **Inventory Control:** Administrators can add new cars, update existing car details (including make, model, year, mileage, and availability), and remove outdated or unavailable vehicles.
   * **Display & Search:** Customers can view a list of available cars along with their relevant specifications.
3. **Rental Booking Process:**
   * **Booking Flow:** Customers are guided through the process of selecting a car, choosing rental dates, and validating the rental period.
   * **Cost Calculation:** The system calculates the rental cost based on car-specific rates and rental duration. It integrates a discount mechanism via loyalty point redemption.
   * **Unique Rental ID:** Each rental request is assigned a unique booking identifier for tracking and future reference.
4. **Payment Processing:**
   * **Simulated Payment Integration:** Payment processing is simulated via an integrated module (e.g., using eSewa as a prototype), and every successful transaction is recorded as a separate payment record.
   * **Data Persistence:** All payment transactions, including amounts and timestamps, are stored in a cloud-based database (Firebase Firestore) to ensure reliability and scalability.
5. **Administrative Oversight:**
   * **Rental Request Approval:** Administrators can review pending rental requests, view customer rental histories and associated payment records, and make decisions to approve or reject bookings.

# Exclusions and Limitations

* **Real Payment Gateway:**

While the payment simulation emulates user confirmation (e.g., through prompts), integration with an actual payment gateway or financial institution is outside the scope of this project.

* **Scalability for High-Volume Transactions:**

This project is designed as a demonstrative solution and academic exercise. Although it utilizes Firestore for data persistence, it is not fully optimized or stress-tested for high-volume, production-level traffic.

* **Advanced Security Measures:**

While basic authentication and data integrity practices are implemented, advanced security features (e.g., multi-factor authentication, encrypted data transmission) are considered beyond the current scope.

* **Mobile Responsiveness or Native Applications:**

The current system is implemented as a desktop or web-based application. Dedicated mobile apps or cross-platform responsiveness have not been developed.

* **Integration with External Systems:**

Integration with third-party services beyond Firebase (e.g., vehicle tracking systems and customer relationship management systems) is not included in the initial release of the project.

* **Multiple Booking of Cars on the Same Dates:**

This system currently does not check if a car is already booked for specific dates and continues to accept new bookings regardless. As a result, multiple bookings may occur on the exact dates. Future iterations should consider implementing a booking conflict management system to prevent overbooking

# Features

1. **User Management:**
   * **User Registration:** Ability for customers and administrators to create new accounts.
   * **Secure Login/Authentication:** Validating user credentials to ensure secure access.
   * **Loyalty Points Tracking:** Displaying available loyalty points and enabling redemption for discounts.
2. **Car Management:**
   * **Add Car:** Admins can add new cars to the fleet, providing details such as make, model, year, mileage, and rental conditions.
   * **List Cars:** Displaying available cars along with their specifications and rental status.
   * **Update Car Information:** Editing existing car details.
   * **Delete Car:** Removing cars that are no longer available.
3. **Rental Booking:**
   * **Rental Request Process:** Guiding users through selecting a car, choosing rental dates, and confirming the rental.
   * **Cost Calculation:** Automatically computing the rental cost based on a base rate duration and applying any loyalty point discounts.
   * **Unique Booking Identification:** Automatically generates a unique booking ID for each rental.
4. **Payment Processing:**
   * **Simulated Payment:** Integration with a simulated eSewa payment module for processing payments.
   * **Recording Payment Transactions:** Storing each payment as a separate record with details like amount and timestamp.
5. **Administrative:**
   * **Rental Approvals:** Allowing administrators to review, approve, or reject pending rental requests.
   * **Customer Records Review:** Providing access to detailed rental histories and payment records for oversight.
   * **Loyalty Points Management:** Automatically updating loyalty points based on rental duration and car bonus points.
6. **System Architecture:**
   * **Modular Design:** Separation of functionalities into distinct modules (user management, car management, booking management, payment processing).

# Installation and Setup

**System Requirements**

* Operating system: Windows
* Python 3.6+ is installed on your system.
* A Valid Firebase project with Firebase enabled.
* The Firebase Admin SDK service account JSON file. (Included in the repository with name -"serviceAccount.Json")
* Required Python packages: firebase-admin

**Configuration**

1. **Clone the Repository:**
   * Open VScode and click on Clone Git Repository..
   * Paste the following code when prompted "git clone https://github.com/Biplab1992/Car\_Rental\_System"
2. **Create a New Virtual Environment:**
   * python -m venv Biplab
   * Biplab\Scripts\activate
3. **Select the Biplab Interpreter:**
   * Press Ctrl+Shift+P to open Command Palette
   * Select " Python(version name)('Biplab':venv)
4. **Install the Firebase Admin SDK**: pip install firebase-admin
5. Run the program via main.py

# Project Files and Their Purpose

1. **main.py**: The entry point of the application. Contains the main loop for handling user interactions and coordinating system processes.
2. **Requirements.txt**: Lists all external packages required to run the project.
3. **ReadMe Files**: Contain settings for database connections, environment variables, and other system configurations.
4. **user\_manager.py**: Handles tasks such as user registration, login, and profile management.
5. **car\_manager.py**: Responsible for operations related to adding, removing, or editing vehicle information.
6. **booking\_manager.py**: Manage the booking workflow, payment processing, and rental agreement generation.
7. **database.py:** To serve as a centralized interface for managing all interactions with the database, including accessing and updating user, car, rental, and payment records using Firebase Firestore.
8. **Payment\_gateway.py**: To simulate the payment processing workflow, acting as a placeholder for real payment integration in the Car Rental System.
9. **serviceAccount.json**: To securely store the private credentials required for initializing the Firebase Admin SDK, enabling authenticated access to Firebase services such as Firestore for the System.

# Diagrams

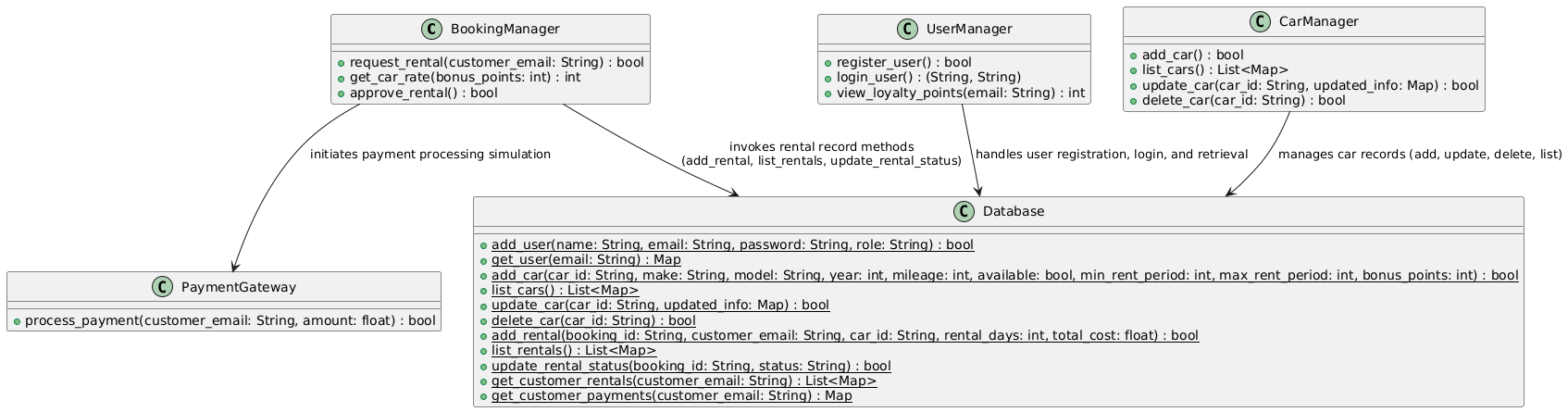
1. Class Diagram

Figure 1: Class diagram of Car Rental System

This diagram illustrates the architecture of a car rental management system, combining core functionalities into five main classes. The UserManager manages user registration, authentication, and loyalty point management, while the CarManager supervises operations such as adding, listing, updating, and deleting car records. The Booking Manager is accountable for processing rental requests, calculating rental costs, and handling rental approvals, effectively coordinating between users and inventory. The PaymentGateway class simulates payment processing to manage financial transactions, and the Database class functions as the central interface for persisting and retrieving data related to users, cars, rentals, and payments from storage. The interactions depicted by the arrows demonstrate how these classes collaborate to provide a consistent, integrated solution for managing the car rental process.

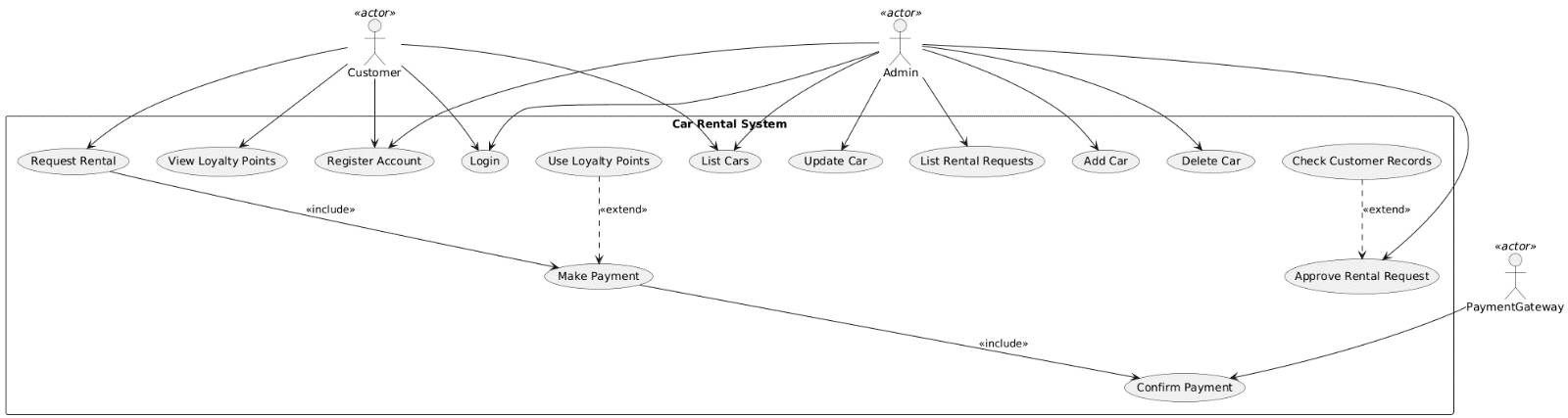
1. Use Case Diagram

Figure 2: Use Case diagram of Car Rental System

This diagram showcases the main functionalities, including user registration and login managed by the UserManager, car inventory operations handled by the CarManager, rental processing and approval through the BookingManager, simulated payment processing via the payment gateway, and persistent data storage performed by the Database. Together, these components enable actors—both customers and administrators seamlessly execute tasks such as booking a car, managing vehicle records, processing payments, and tracking rental histories, illustrating a cohesive and integrated system architecture essential for efficient rental operations.

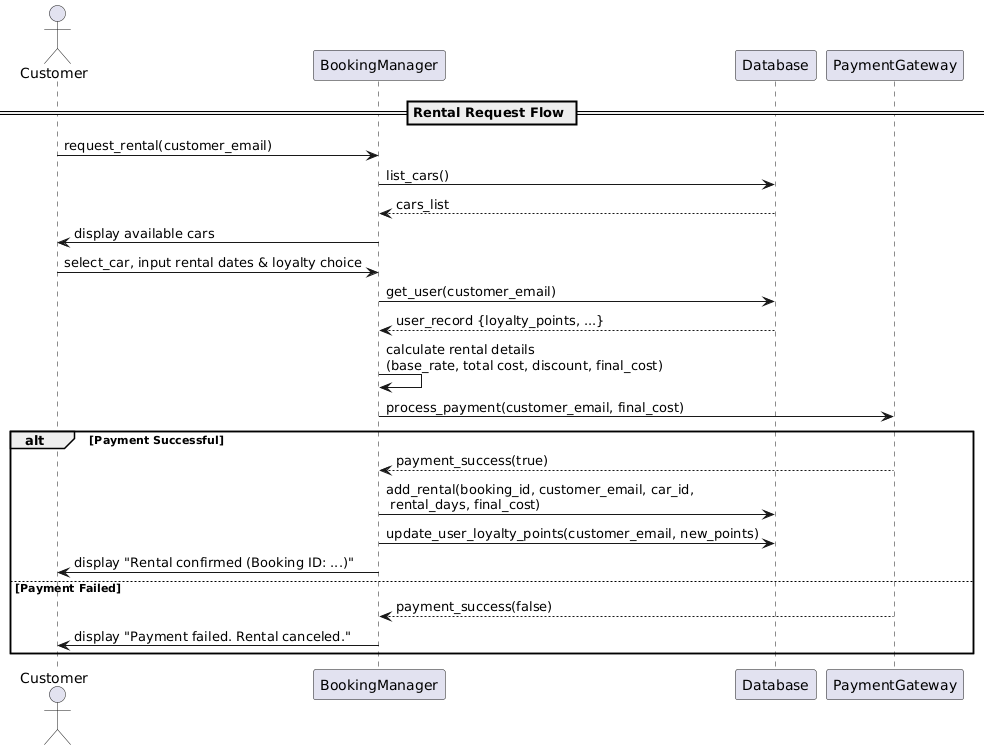
1. Sequence Diagram

Figure 3: Sequence diagram of Car Rental System

When a customer initiates a rental request, the process begins with the **Booking Manager** receiving the customer's selected car and rental period details. The **Booking Manager** first verifies the user's credentials by consulting the **User Manager** and checking the customer's loyalty points for any applicable discounts. Simultaneously, it consults the **Car Manager** to confirm that the chosen vehicle is available. Once these preliminary validations are complete, the **Booking Manager** computes the rental cost using the car's rate and the rental duration, applying any loyalty-based discounts if applicable. The computed payment details are then forwarded to the **payment gateway**, which simulates the payment processing and returns a confirmation of success. Following payment confirmation, the **Booking Manager** updates the system by recording the rental and payment details in the **database. This** ensures that the booking is formally registered, the car's availability is updated, and the customer's loyalty points are adjusted accordingly. Finally, the system generates a unique rental ID and sends a booking confirmation to the customer, completing the transaction.

1. UML Diagram

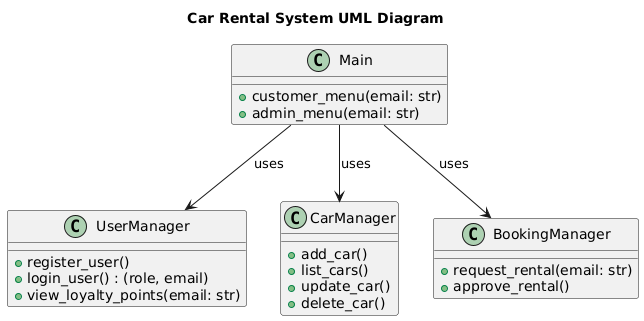


Figure 4: Unified Modeling Language (UML) diagram of Car Rental System

This diagram shows the main module—which encapsulates functions like customer\_menu, and admin\_menu which highlights its associations with three key classes: **UserManager** (handling user registration, login, and loyalty points), **CarManager** (managing car inventory through adding, listing, updating, and deleting cars), and **BookingManager** (processing rental requests and approving rentals). This diagram concisely illustrates how the system's user-facing module utilizes these specialized classes to perform essential operations, highlighting the separation of responsibilities.

# License

MIT License

Copyright (c) 2025 Biplab1992

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# Development

The development of the Car Rental System project follows a structured and iterative software engineering approach that emphasizes modularity, maintainability, and scalability. The project is primarily developed in **Python** (version 3.13.3) and leverages the **Firebase Admin SDK** for interfacing with Firebase Firestore, ensuring robust and cloud-based data management. The application is designed using object-oriented principles with distinct modules handling user management, car inventory, rental processing, payment simulation, and database interactions.

**Language and Frameworks**

* **Primary Language:** Python
* **Backend and Database Integration:** The Firebase Admin SDK for Python provides seamless authentication and interaction with Firestore, serving as the cloud-based database.
* **Design Patterns:** The system is structured around modular design patterns, ensuring that functionalities are well encapsulated and isolated, facilitating future enhancements and maintenance.

**Process**

* **Planning and Requirements Gathering:**

The project began with a comprehensive analysis of user needs and system requirements, defined by creating use case diagrams, class diagrams, and flowcharts to capture the core functionalities, including user registration, car management, booking, and payment processing.

* **System Design:**

The design phase involved developing a modular architecture with clearly separated modules for each core functionality. UML diagrams were created to visualize the system structure, ensuring clarity on how individual modules interact through defined interfaces.

* **Implementation:**

The code is structured into several modules: **UserManager, CarManager, BookingManager, PaymentGateway**, and **Database,** each handling specific aspects of the car rental process. Developers utilize the **Biplab** virtual environment to ensure dependency isolation and consistent configuration across development machines. Features are implemented incrementally following best coding practices.

* **Deployment and Future Enhancements:**

Upon successful testing and code review, the system is deployed for demonstration purposes, accompanied by relevant documentation. Future development plans include integrating a real payment gateway, enhancing mobile responsiveness, and refining security protocols.

# Tools

1. **Integrated Development Environment (IDE):**

Visual Studio Code is used for writing, debugging, and managing application code.

1. **Version Control:**

GitHub is employed for tracking code changes and collaborating with remote repositories.

1. **Virtual Environment:**

Python's built-in venv module is used to create an isolated development environment.

1. **Diagram Generation:**

Draw.io and PlantUML is used to design and generate UML diagrams, which provide clear visual representations of the system's architecture.

1. **Database Management:**

Firebase (specifically Firestore, integrated via the Firebase Admin SDK) is used as the cloud-based database, handling data storage for users, rentals, and transactions.

# Future Maintenance and Support:

The project is designed with long-term maintainability in mind. Future maintenance will include regular code reviews, automated testing (utilizing tools such as Pytest), and continuous integration processes to minimize defects. Documentation will be continuously updated, and detailed changelogs, along with deprecation notices, will be provided to support developers and end-users during upgrades. Active community engagement will also help in rapidly addressing bugs and incorporating feature requests.

**Versioning:**

We plan to follow Semantic Versioning, where every release is denoted by a version number in the format MAJOR.MINOR.PATCH.

* MAJOR releases signify breaking changes that may require adjustments from users.
* MINOR releases add new features in a backward-compatible manner.
* PATCH releases include bug fixes and minor improvements that do not impact user workflows.

**Backward Compatibility:**

Backward compatibility is a fundamental design consideration that safeguards new updates from disrupting existing functionality. This is achieved by designing APIs and public interfaces with future expansions in mind, incorporating tests that validate old behaviors against new code, and providing migration guides when breaking changes are unavoidable. Deprecated functions or features will be maintained temporarily with clear warnings, allowing users time to adapt before their eventual removal.

# Conclusion

In conclusion, the Car Rental System project showcases a modern and modular approach to software design and development. It utilizes powerful tools, including Visual Studio Code for development, Firebase for cloud-based data management, and Draw.io for creating clear architectural diagrams. The project is built with scalability and maintainability as primary goals.

Rigorous version control is maintained using Git, and we adhere to Semantic Versioning to ensure that each release is systematic and backward-compatible. This approach minimizes disruption for existing users while preparing the system for future enhancements.

Overall, this project not only meets current use-case requirements but also establishes a strong foundation for iterative improvements. It reflects a commitment to quality, collaboration, and long-term support in software engineering.

# ReadMe

**Installation and Configuration**

1. **Clone the Repository:**

* Open VScode and click on Clone Git Repository.
* Paste the following code when prompted "git clone <https://github.com/Biplab1992/Car_Rental_System>"

1. **Create a New Virtual Environment:**

* python -m venv Biplab
* Biplab\Scripts\activate

1. **Select the Biplab Interpreter:**

* Press Ctrl+Shift+P to open Command Palette
* Select " Python(version name)('Biplab':venv)

1. **Install the Firebase Admin SDK: pip install firebase-admin**

**Running the System**

To start the Car Rental System, simply run: python main.py. This will display the main menu with option as follows:

1. **Register User:** New user can create an account by providing their name, email id, password and role.
2. **Login:** Admins and customers have different menus where Admins can access the features like Add, update, delete cars, approve rental requests, list cars while customers can see available cars and request for rentals, view their loyalty points.
3. **Exit:** Choose to close the application.

**Credits**

Developer: Biplab Neupane

(Auckland, New Zealand)

neubiplab@gmail.com

Feel free to contact me for any questions, support or further development inquiries.

# References

Documentation, G. (n.d.). *Git Documentation*. Retrieved from https://git-scm.com/doc

Draw.io. (n.d.). *Draw.io Documentation*. Retrieved from https://app.diagrams.net/

Firebase. (n.d.). *Cloud Firestore Documentation*. Retrieved from https://firebase.google.com/docs/firestore

Microsoft. (n.d.). *Visual Studio Code Documentation*. Retrieved from https://code.visualstudio.com/docs

PlantUML. (n.d.). *PlantUML Documentation*. Retrieved from https://plantuml.com