Shape

Description automatically generated with medium confidence

EEET2482/EEET2653 Software Engineering Design

COSC2082/COSC2721 Advanced Programming Techniques

Semester 2, 2025

Group Project Report

**E-MOTORBIKE RENTAL APPLICATION**

**Lecturer:** Dr Tri Huynh

**Group No. 5**

**Group Members:**

Lu Duc Thinh (S3992133)

Jang Soohyuk (S3928379)

Pham Tuan Hai (S3975144)

Vu The Quyen (S4027077)

**Date** : 18/09/2025

# I. INTRODUCTION

This project is about developing a peer-to-peer electric motorbike rental application that allows users to share their personal motorbikes with other community members. The system works like a sharing platform where people can list their motorbikes for rent and book motorbikes from others using credit points.

The application supports three types of users: Guests who can browse basic motorbike information, Members who can list and rent motorbikes, and Administrators who manage the system. Members can register their motorbikes with rental details, search for available bikes by date and location, make booking requests, and rate each other after completed rentals.

Key features include user authentication and registration, motorbike listing management, search and filtering based on user eligibility, booking approval system, credit point transactions, and mutual rating system. The system also enforces various constraints such as license requirements for larger motorbikes and credit balance management.

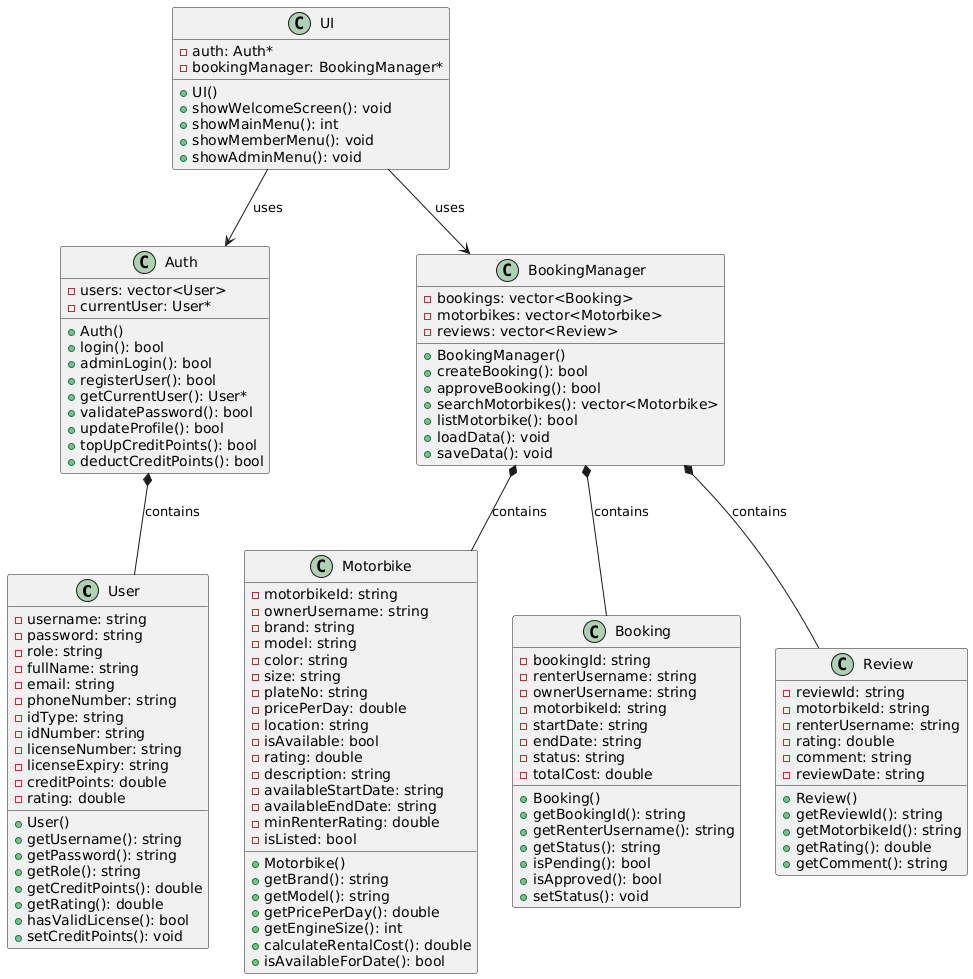
The project demonstrates object-oriented programming concepts such as class inheritance and encapsulation through a modular file structure.

# II. APPLICATION DESIGN AND DEVELOPMENT

## **1. Software Design (Class Diagram)**

*From the application description, provide class diagram and description/ explanation as below:*

Class Diagram:



*Figure 1: Class diagram for E-motorbike rental application*

Description of Each Class:

|  |  |  |  |
| --- | --- | --- | --- |
| Class: User | Name and Data type | Description | Reason/Explanation (why we need it as an attribute/ method) |
| Attribute(s) | username: string | Unique identifier for user login | Need unique ID to distinguish users and prevent duplicate accounts |
|  | password: string | User's login password | Essential for authentication and account security |
|  | role: string | User role (member/admin) | Different roles have different access permissions in the system |
|  | fullName: string | User's complete name | Required for registration and profile identification |
|  | email: string | User's email address | Contact information and account verification |
|  | phoneNumber: string | User's phone number | Contact information for communication |
|  | idType: string | ID document type (Citizen ID/Passport) | Required for user identity verification process |
|  | idNumber: string | ID document number | Unique identification for verification |
|  | licenseNumber: string | Driver's license number | Required to validate license for motorbikes >50cc |
|  | licenseExpiry: string | License expiration date | Check if license is still valid for rental eligibility |
|  | creditPoints: double | User's credit balance | Currency for rental transactions in the system |
|  | rating: double | User's renter rating | Trust system - other users rate renter behavior |
| Method(s) | getUsername(): string | Returns user's username | Need to access username for login and identification |
|  | getCreditPoints(): double | Returns current credit balance | Essential for checking rental affordability |
|  | hasValidLicense(): bool | Checks if license is valid | Enforces 50cc+ motorbike rental restrictions |
|  | setCreditPoints(): void | Updates credit balance | Required for top-ups and rental payments |

*Table 1: Description table for User class*

|  |  |  |  |
| --- | --- | --- | --- |
| Class: Auth | Name and Data type | Description | Reason/Explanation (why we need it as an attribute/ method) |
| Attribute(s) | users: vector<User> | Collection of all registered users | Store all user accounts in the system |
|  | currentUser: User\* | Pointer to currently logged-in user | Track active session for personalized features |
| Method(s) | login(): bool | Authenticate member login | Verify credentials and establish user session |
|  | adminLogin(): bool | Authenticate admin login | Separate admin authentication for security |
|  | registerUser(): bool | Create new user account | Allow new members to join the platform |
|  | validatePassword(): bool | Check password strength | Enforce strong password policy as required |
|  | topUpCreditPoints(): bool | Add credits to user account | Enable users to purchase rental currency |
|  | deductCreditPoints(): bool | Remove credits from account | Process rental payments |

*Table 2: Description table for Auth class*

|  |  |  |  |
| --- | --- | --- | --- |
| Class: Motorbike | Name and Data type | Description | Reason/Explanation (why we need it as an attribute/ method) |
| Attribute(s) | motorbikeId: string | Unique motorbike identifier | Distinguish between different motorbikes |
|  | ownerUsername: string | Username of motorbike owner | Link motorbike to its owner for rental requests |
|  | brand: string | Motorbike brand (VinFast, Honda, etc.) | Important for user selection and preferences |
|  | model: string | Motorbike model name | Specific model information for rental decisions |
|  | size: string | Engine size (50cc, 125cc, etc.) | Determines license requirements and pricing |
|  | pricePerDay: double | Daily rental rate in CP | Owner sets pricing for their motorbike |
|  | location: string | City location (HCMC/Hanoi) | Geographic filtering for search functionality |
|  | availableStartDate: string | Rental period start date | Define when motorbike becomes available |
|  | availableEndDate: string | Rental period end date | Define rental availability window |
|  | minRenterRating: double | Minimum required renter rating | Owner sets quality standards for renters |
| Method(s) | getBrand(): string | Returns motorbike brand | Display brand information in listings |
|  | getPricePerDay(): double | Returns daily rental rate | Calculate total rental costs |
|  | getEngineSize(): int | Extract engine size from size string | Determine license requirements (50cc rule) |
|  | isAvailableForDate(): bool | Check date availability | Validate booking requests against availability |

*Table 3: Description table for Motorbike class*

|  |  |  |  |
| --- | --- | --- | --- |
| Class: Booking | Name and Data type | Description | Reason/Explanation (why we need it as an attribute/ method) |
| Attribute(s) | bookingId: string | Unique booking identifier | Track individual rental transactions |
|  | renterUsername: string | Username of person renting | Link booking to renter for management |
|  | ownerUsername: string | Username of motorbike owner | Link booking to owner for approval |
|  | motorbikeId: string | ID of motorbike being rented | Connect booking to specific motorbike |
|  | startDate: string | Rental start date | Define rental period beginning |
|  | endDate: string | Rental end date | Define rental period end |
|  | status: string | Booking status (Pending/Approved/Rejected) | Track booking lifecycle |
|  | totalCost: double | Total rental cost in CP | Store calculated payment amount |
| Method(s) | getStatus(): string | Returns current booking status | Check booking state for UI display |
|  | isPending(): bool | Check if booking awaits approval | Determine if owner action is needed |
|  | isApproved(): bool | Check if booking is confirmed | Identify active rentals |
|  | setStatus(): void | Update booking status | Change status during approval process |

*Table 4: Description table for Booking class*

|  |  |  |  |
| --- | --- | --- | --- |
| Class: Review | Name and Data type | Description | Reason/Explanation (why we need it as an attribute/ method) |
| Attribute(s) | reviewId: string | Unique review identifier | Track individual reviews |
|  | motorbikeId: string | ID of reviewed motorbike | Link review to specific motorbike |
|  | renterUsername: string | Username of reviewer | Identify who wrote the review |
|  | rating: double | Numerical rating (1-5) | Quantified feedback for motorbike quality |
|  | comment: string | Written review comment | Detailed feedback for other users |
|  | reviewDate: string | Date when review was written | Track review timeline |
| Method(s) | getComment(): string | Returns review comment | Display customer feedback to potential renters |
|  | getRating(): double | Returns numerical rating | Calculate average motorbike ratings |

*Table 5: Description table for Review class*

|  |  |  |  |
| --- | --- | --- | --- |
| Class: BookingManager | Name and Data type | Description | Reason/Explanation (why we need it as an attribute/ method) |
| Attribute(s) | bookings: vector<Booking> | Collection of all bookings | Manage all rental transactions |
|  | motorbikes: vector<Motorbike> | Collection of all motorbikes | Store all available motorbikes |
|  | reviews: vector<Review> | Collection of all reviews | Store customer feedback data |
| Method(s) | createBooking(): bool | Create new rental request | Allow members to request motorbike rentals |
|  | approveBooking(): bool | Approve pending booking | Let owners confirm rental requests |
|  | searchMotorbikes(): vector<Motorbike> | Find available motorbikes | Help users discover suitable rentals |
|  | loadData(): void | Load data from files | Restore system state on startup |
|  | saveData(): void | Save data to files | Persist data when application closes |

*Table 6: Description table for BookingManager class*

|  |  |  |  |
| --- | --- | --- | --- |
| Class: UI | Name and Data type | Description | Reason/Explanation (why we need it as an attribute/ method) |
| Attribute(s) | auth: Auth\* | Pointer to authentication system | Access user login and authentication features |
|  | bookingManager: BookingManager\* | Pointer to booking system | Access motorbike and booking functionality |
| Method(s) | showWelcomeScreen(): void | Display application intro | Provide user-friendly application startup |
|  | showMainMenu(): int | Display main menu options | Allow users to choose their role and actions |
|  | showMemberMenu(): void | Display member-specific options | Provide member functionality access |
|  | showAdminMenu(): void | Display admin-specific options | Provide administrative tools access |

*Table 7: Description table for UI class*

Class Relationships:

• Auth is a Composition part of User:

The Auth class contains and manages a collection of User objects through a vector<User>. Auth owns the lifecycle of User objects - when Auth creates a new user through registration, it adds the User object to its collection. When Auth is destroyed, all User objects are also destroyed. This composition relationship ensures centralized user management and authentication control.

• BookingManager is a Composition part of Booking:

BookingManager contains and owns all Booking objects in the system. It creates new Booking objects when users make rental requests, manages their status changes (pending, approved, rejected), and handles their persistence to files. The BookingManager controls the complete lifecycle of booking transactions and ensures business rules are enforced.

• BookingManager is a Composition part of Motorbike:

BookingManager owns and manages all Motorbike objects in the system. It handles motorbike registration when users list their vehicles, manages search and filtering operations, and controls motorbike availability status. This composition allows centralized motorbike management and ensures data consistency across the rental system.

• BookingManager is a Composition part of Review:

BookingManager contains all Review objects and manages the review system. It loads reviews from files, provides review data for motorbike listings, and handles review-related operations. This composition ensures that reviews are properly managed as part of the overall booking and rental system.

• UI depends on Auth:

The UI class has an association relationship with Auth, using it for user authentication and session management. UI calls Auth methods to handle login processes, user registration, and access to current user information. This dependency allows UI to provide authentication features without implementing the authentication logic itself.

• UI depends on BookingManager:

The UI class depends on BookingManager to access all motorbike and booking functionality. UI delegates business operations like motorbike search, booking creation, and rental management to BookingManager. This association maintains separation of concerns where UI handles user interaction while BookingManager handles business logic.

## **2. Implementation Result**

**a. Welcome Screen**

Result Summary:

The welcome screen successfully displays project information including group members, instructors, and main menu options. Users can choose between five options: Guest access for limited browsing, Member login for full functionality, Admin access for system management, new member registration, or exit to terminate the application.

Screenshots of Sample Result:

A screenshot of a computer program

AI-generated content may be incorrect. *Figure 2: Welcome screen displaying project information and main menu options*

**b. Basic Features**

Result Summary:

|  |  |  |  |
| --- | --- | --- | --- |
| Feature Name | Feature Description | Status (Implemented/Not Implemented) | Bugs/Limitations if it has |
| 1. Guest access | Guests can browse general motorbike listings with limited details: brand, model, engine size and location. | Implemented | No major limitations. Works as designed with proper access restrictions. |
| 2. Admin access | Admin can view all member profiles and motorbike listings without restriction. | Implemented | No limitations. Admin has full system access and can view comprehensive statistics. |
| 3. Member registration | Users can register as members and register one electric motorbike. Registration must include a strong password policy. Upon successful registration and payment, the member receives 20 credit points and a default renter rating of 3. All members and motorbike information must be recorded and stored. | Implemented | Registration fee payment is simulated (not real payment processing). Motorbike registration during signup is separate menu option rather than integrated. |
| 4. Profile and account management | Members can log in with their username and password. They can view and update their profile information as specified in the Profile management section, change their password, and top up their credit points. | Implemented | Credit point top-up is simulated (not real payment gateway). Username cannot be changed for system integrity. |
| 5. Motorbike listing | Members can list an electric motorbike as specified in the Motorbike listing section. | Implemented | No major limitations. System enforces one motorbike per member rule effectively. |
| 6. Motorbike unlisting | Members can un-list their motorbike at any time, unless it is already booked. | Implemented | No limitations. System properly checks for active bookings before allowing unlisting. |
| 7. Motorbike search and filtering | Members can search for available motorbikes by date and city. Search results should follow the filtering criteria defined in the Motorbike search and filtering section. | Implemented | Date validation uses simple string comparison rather than proper date parsing. Works correctly for DD/MM/YYYY format. |
| 8. Viewing listings and reviews | Members can view complete listings, including the average rating score and user comments for each motorbike. | Implemented | Reviews are loaded from file rather than dynamically created. Review writing functionality not implemented (read-only). |
| 9. Rental request submission | Members can submit rental requests. A license restriction must be enforced: members without a valid motorbike license cannot rent electric motorbikes over 50cc. | Implemented | License validation uses simple date comparison. More sophisticated date parsing could improve accuracy. |
| 10. Rental request management | Motorbike owners can view all rental requests for their listed motorbike and choose to accept one. All overlapping rental requests are automatically rejected upon acceptance. | Implemented | No limitations. Automatic rejection of overlapping requests works correctly. |
| 11. Rental confirmation and credit deduction | When a request is accepted, CPs are deducted from the renter's balance, and the motorbike is marked as rented for the selected period. Rental cancellations are not allowed once confirmed. | Implemented | No limitations. Credit deduction and rental period management work as designed. |
| 12. Ride completion and ratings | After the rental period ends, both the renter and the owner must rate each other (1-5 stars and a comment). Ratings and comments are tied to the specific transaction. | Implemented | Mutual rating system implemented. Rating submission works correctly for completed rentals. |
| 13. Rating aggregation | Member and motorbike ratings are automatically averaged over time and updated accordingly. | Implemented | Simple averaging algorithm used. More sophisticated weighted averaging could be implemented for improvement. |
| 14. Data persistence | All data must be saved to data file(s) before the application closes. Upon starting the application, the data must be loaded and made available for continued use. | Implemented | File-based storage works reliably. Concurrent access not supported (single-user application design). |

Screenshots of Sample Result (for each feature):

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 3: Guest motorbike listings showing only limited details (Feature 1)*

A computer screen with text

AI-generated content may be incorrect.

*Figure 4: Admin view showing all member profiles without restrictions (Feature 2)*

*A screen shot of a computer

AI-generated content may be incorrect.*

*Figure 5: Admin view showing all motorbikes without restrictions (Feature 2)*

*A screenshot of a computer screen

AI-generated content may be incorrect.*

*Figure 6: Strong password policy during registration (Feature 3)*

*A screenshot of a computer program

AI-generated content may be incorrect.*

*Figure 7: Registration completion with 20 credit points and default rating 3.0 (Feature 3)*

A screenshot of a computer program

AI-generated content may be incorrect.

*Figure 8: E-motorbike registration completion (Feature 3, 5)*

A screenshot of a computer

AI-generated content may be incorrect.

*Figure 9: Member profile information display (Feature 4)*

*A screen shot of a computer screen

AI-generated content may be incorrect.*

*Figure 10: Credit point top-up with password authentication (Feature 4)*

*A screenshot of a computer screen

AI-generated content may be incorrect.*

*Figure 11: Motorbike unlisting confirmation with booking status check (Feature 6)*

**c. Advanced Features**

Result Summary:

Screenshots of Sample Result:

# IV. DISCUSSIONS & CONCLUSIONS

* Short conclusions on the final results.
* Does the implementation match with the design (class diagram)? Do you find it helpful when having a design before and during the implementation? Do you update the design along the way when you do implementation (writing the code)?
* Last but not least, shortly reflect on what you have learnt through this group project and how it can benefit your future study/career.

# V. REFERENCES (USE IEEE STYLES)