Project Title: Car Rental Management System

Course code: CPS2232

Course Name: DATA STRUCTURE **Session**: W03

Student Name: Mi Yixuan Student Number: 1307943

Student Name: Yu Yiduo Student Number: 1306057

Student Name: Zhao Yiyi Student Number: 1305974

Student Name: Yu Qiyang Student Number: 1306031

Student Name: Jia Taoyin Student Number: 1306194

Professor: Dr Ken Ehimwenma, Ph.D.

Date: December 10th, 2024

Table of Contents

Abstract

With the growing travel demand, car rental has an increasingly more significant impact on people's lives. People urgently need a safe, convenient, and stable car rental system software to serve their daily lives. For this reason, this project designs a car rental management system (CRMS) to solve the complex management needs in the vehicle rental industry and provide customers with an efficient and convenient car rental tool. The car rental management system caters to two different user roles-car rental companies and customers and provides them with a friendly user experience. This system allows users to efficiently manage vehicle information, rental transactions, and user registration, providing a modular and scalable solution.

Car rental companies, as administrators, can perform tasks such as importing and managing vehicle data, viewing logs, and maintaining user accounts. This ensures optimal resource allocation and business supervision. On the other hand, customers can register an account and browse available vehicles by attributes such as brand and price after logging in through authentication and initiate rental or return transactions. In addition, the system also implements automatic notifications through email to ensure the user's right to know and maintains transaction logs to enhance traceability.

The CRMS project uses data structures such as ArrayList to implement dynamic data management and retrieval of vehicles and management of user data and integrates file processing for persistent storage. To implement the sending mail function, we use Javax Mail API to send rental confirmation emails. The notification system sends notifications to users about account creation and rental transactions, thereby enhancing communication and service reliability. The system uses a basic role-based access control mechanism to distinguish between administrative functions and customer functions, ensuring that different users can use the system efficiently at the same time.

Challenges addressed during development include optimizing file-based storage, ensuring secure

authentication, and integrating efficient logging mechanisms. Expected results include improved operational efficiency, customer satisfaction, and resource utilization. By modernizing the car rental process through this system, the project aims to demonstrate how digital solutions can change the traditional industry and promote better car rental services. At the same time, the system's modular structure allows for future expansion and improvement.

I. Introduction

In modern car rental services, effective resource management and seamless interaction with users are essential for operational success. The "Car Rental Management System" provides a comprehensive platform to meet these requirements, which helps manage vehicle information, distinguish different user account functions and improve the efficiency of rental transactions. At the same time, the project meets the needs of both rental companies and ordinary users, ensuring that the car rental system is efficient and user-friendly.

- **Functional Overview:**

The system is divided into two main user roles: rental companies and car rental customers. Each role has specific functions to meet their needs:

1. **Rental Company Function:**

Rental companies can act as administrators of this system, with the functions of vehicle management, viewing vehicles and recording rental records

2. **Vehicle Management:**

Add new vehicles to the system, specifying details such as brand, model, color, capacity, rental price and size (length and width). Vehicles are automatically assigned unique IDs and marked as available for rent.

They can also remove vehicles from the system by providing the vehicle ID, ensuring that the database is kept up to date.

3. **View Vehicles:**

Administrators can view a complete list of all vehicles in the system, presented in a table format with all relevant details.

4. **Rental Log:**

Administrators can access a detailed rental log, providing an overview of all past transactions for monitoring and accountability.

5. **Car Rental Customer Functionality:**

Car rental customers as system users have the functions of user registration authentication, query vehicle information, rent vehicles, and return vehicles.

6. **User Registration and Authentication:**

Users can register an account with a valid email and password. The role (admin or user) is determined when creating an account, and the admin account requires additional verification. At the same time, secure authentication ensures that only authorized users can access the system.

7. **Vehicle Browsing and Details:**

Users can browse the list of available vehicles or view the details of a specific vehicle by providing its ID.

8. **Renting Vehicle:**

Users can rent a vehicle by specifying a rental period. The system marks the vehicle as unavailable during the rental period and notifies the user of the rental confirmation via email.

9. **Returning Vehicle:**

After returning the rented vehicle, the system updates the availability of the vehicle and adjusts the user's rental record accordingly.

- **Main Features**

1. **Persistent Data Management:**

Vehicle Data: Vehicle details are stored in text files and loaded into memory at runtime. Any changes, such as adding or removing vehicles or updating their availability, are saved back to the file.

User Accounts: User registration information, including roles and rental history, is stored in separate files, enabling persistent and secure account management.

2. **Email Notification System:**

The system integrates JavaMail API to send email notifications. For example, users will receive

a confirmation email after successfully renting a vehicle, ensuring timely and effective notifications.

3. **Rental Log and Tracking:**

All transactions are recorded in a dedicated file, providing a history of rental activities for management review. This feature improves transparency and accountability.

4. **Role-based Menus and Actions:**

The system provides different menus and functions for administrators and ordinary users, ensuring intuitive navigation and access to role-specific functions.

5. **User-friendly Data Presentation:**

Vehicle information is displayed in a well-structured table format for easy viewing, including attributes such as ID, brand, model, color, capacity, rental, size, and availability.

6. **Error Handling and Notification:**

The system includes powerful error handling capabilities, such as notifying users when trying to rent an unavailable vehicle or providing invalid input.

- **Innovation and Contribution**

During the development of the car rental management system, several challenges were encountered, which we solved through innovative programming techniques:

1. **Data Storage and Synchronization:**

Challenge: Balancing persistent storage and fast in-memory data access and ensuring synchronization of updates between the two.

Solution: The system uses a hybrid approach where vehicle and user data are loaded from files into memory for fast access and written back after each modification.

2. **Email Integration:**

Challenge: Implement a reliable and secure email notification system.

Solution: Use the JavaMail API and optimize the configuration to ensure compatibility with

SMTP servers. This feature ensures that users receive updates in a timely manner.

3. **Log Management:**

Challenge: Provide a transparent log of rental activity while minimizing performance overhead.

Solution: A separate logging mechanism records system activity in a dedicated file that administrators can review at any time.

4. **Role-Based Access Control:**

Challenge: Ensure that the system can distinguish between administrator and normal user roles without compromising usability.

Solution: Implement role-based menus and perform specific checks during login to determine the user's role and display the appropriate interface.

- **Presentation Overview:**

01 Introduction

1. Purpose
2. Key Features

02 CarRental ManagementSystem class

1. Initializes the manager and takes care of user login and registration
2. Function menus are differentiated according to user identity
3. Invokes functional modules related to administrators or users

03 RentalManager class

1. Build the data structure and construction method of the lease
2. Introduce the javax.mail package to implement the email sending function.
3. The data structure is mainly based on ArrayList.

04 User class

1. The User class represents a user object with key attributes: email, password, a boolean isAdmin to indicate if the user is an administrator, and numOfRentCars to track the number

of cars rented by the user. It encapsulates user-related data securely.

2. The class provides a constructor for initializing user instances and includes getter and setter methods to access and modify its properties, ensuring flexibility and controlled access.
3. The isAdmin attribute differentiates administrators from regular users. The toString() method offers a concise representation of user information (email and admin status) for debugging and output purposes.

05 Vehicle class

1. The Vehicle class represents a vehicle object, encapsulating key attributes such as id, brand, model, color, peopleCapacity, rentPrice, dimensions (length and width), and its availability status (isAvailable). It is designed to manage vehicle-related data effectively.
2. The setAvailable method updates the vehicle's availability status. When the vehicle is marked unavailable (isAvailable equals "false"), the details are logged to a text file (Vehicles.txt) for record-keeping. This demonstrates an integration of data updates with file handling.
3. The toString() method provides a detailed string representation of the vehicle's attributes, including brand, model, capacity, rental price, dimensions, and availability status. This facilitates debugging and improves readability when displaying vehicle information.

06 Conclusion

1. System Overview
2. Key Achievements
3. Future Outlook

07 Reference

II. Related Works

Mrena et al. examine the performance benefits of array-based lists like ArrayList over linked lists in their experimental comparison of list implementations. They find that ArrayList leverages CPU caching effectively due to its contiguous memory allocation, resulting in significant performance gains during sequential and random-access scenarios. The study highlights that ArrayList's constant-time O(1) access to elements makes it particularly suitable for scenarios requiring frequent element retrieval by index, outperforming linked lists' linear-time O(n) access .

Furthermore, the paper notes that while the worst-case time complexity of insertion and removal operations for ArrayList can be linear, its amortized complexity remains constant if an efficient expansion strategy is used. This advantage is particularly evident when pre-allocating capacity, allowing for smoother handling of dynamic size adjustments. In contrast, the fragmented memory allocation of linked lists leads to poor cache utilization and slower traversal speeds, especially for large datasets.

By presenting experimental data across various scenarios, Mrena et al. validate that the design of ArrayList offers a versatile and efficient solution for developers. Choosing ArrayList over linked lists can lead to significant performance improvements in applications emphasizing random access or iterative operations. (Mrena et al., 2022)

According to Jacobson et al., with the release of Java 5.0, the ArrayList structure is more conceptually and implementation-aware than arrays for representing contiguous lists. In the long run, this shift can lead to more efficient and effective programming practices. (Jacobson & Thornton, 2004)

III. Research Methodology

The car rental management system is built around four major subsystems: 1) Menu and User Interaction System, 2) Vehicle System, 3) Transaction and Log Management System, 4) User System. These systems are supported by various auxiliary functions to ensure seamless data handling and user experience. Here is a detailed breakdown of the methods used in the implementation:

```
1,Toyota,Corolla,White,5,50.0,4.5,1.8,true
2,Honda,Civic,Black,5,60.0,4.7,1.9,true
3,Ford,Focus,Blue,5,55.0,4.6,1.8,true
4,Nissan,Altima,Gray,5,70.0,4.9,1.9,true
5,Chevrolet,Malibu,Red,5,65.0,4.8,1.9,true
6,BMW,3 Series,White,5,100.0,4.7,1.8,true
7,Mercedes-Benz,C-Class,Black,5,120.0,4.8,1.9,true
8,Audi,A4,Silver,5,110.0,4.7,1.9,true
9,Volkswagen,Passat,Blue,5,75.0,4.9,1.9,true
10,Hyundai,Elantra,Gray,5,50.0,4.5,1.8,true
11,Tesla,Model 3,White,5,130.0,4.7,1.8,true
12,Tesla,Model Y,Black,7,150.0,4.8,2.0,true
13,Toyota,RAV4,Silver,5,80.0,4.6,1.9,true
14,Ford,Escape,Blue,5,85.0,4.6,1.9,true
15,Jeep,Wrangler,Red,5,90.0,4.7,2.1,true
16,Kia,Sorento,Gray,7,70.0,4.8,1.9,true
17,Hyundai,Tucson,White,5,60.0,4.5,1.8,true
18,Honda,CR-V,Black,5,75.0,4.6,1.9,true
19,Mazda,CX-5,Blue,5,70.0,4.5,1.8,true
20,Subaru,Outback,Green,5,75.0,4.8,1.9,true
21,Volvo,XC60,Silver,5,100.0,4.7,1.9,true
22,Lexus,RX 350,Gray,5,110.0,4.7,1.9,true
23,Chevrolet,Traverse,White,8,90.0,5.0,2.0,true
24,Nissan,Rogue,Black,5,70.0,4.6,1.9,true
25,BMW,X5,Blue,5,150.0,4.9,2.0,true
26,Audi,Q5,Silver,5,130.0,4.8,1.9,true
27,Mercedes-Benz,GLC,Black,5,140.0,4.8,1.9,true
28,Volkswagen,Tiguan,Blue,7,75.0,4.7,1.9,true
29,Toyota,Highlander,Red,8,120.0,4.9,2.0,true
30,Honda,Pilot,Gray,8,110.0,4.9,2.0,true
```

Table1: Simple Data

- **Menu and User Interaction System: Car Rental Management System**

1. **User Registration and Authentication:**

The User class encapsulates user information such as email, password, role (admin or user), and number of rented vehicles. The system supports user registration, where data is persistently stored in a text file (Accounts.txt). An ArrayList structure is used to map emails to user objects for quick access at login.

2. **Menu System:**

The menu is role-based and provides different options for administrators and regular users. It employs multiple while loops and switch statements to process user input and navigate through

system functions. Error messages are triggered when users enter invalid instructions, and exception handling guides users to provide the correct information.

- **Vehicle System: Vehicle**

1. **Data Source:**

Vehicle data is stored in a structured text file (Vehicles.txt), where each row represents a vehicle and each column represents a specific attribute, such as brand, model, capacity, rental, size, and availability.

Vehicle Class: Define a Vehicle class to represent each vehicle in the system. It includes attributes such as ID, brand, model, color, capacity, rental, size (length and width), and availability status.

2. **Reading and storing data:**

When the system is initialized, FileReader reads vehicle data from the file line by line.

For each data, a new Vehicle object is created and added to ArrayList<Vehicle>. This ensures that all vehicles are loaded into memory for efficient processing. At the same time, any changes made by the user to the vehicle data are synchronized back to the file using the saveVehicles method.

- **Transaction and Log Management System: RentalManager**

1. **Vehicle Transaction:**

Users can browse available vehicles by searching based on attributes such as brand, model, or price. After selecting a vehicle, they can rent or return it. After renting a vehicle, its availability status is updated, and the user receives a confirmation email. The system does not remove the vehicle from storage but updates its availability and related user information, thus maintaining efficient inventory management. Rental records are recorded in the user's transaction history, and administrators can view these logs for monitoring.

2. **Log management:**

Each system activity (such as user login, vehicle rental, or return) is recorded in a log file (Log.txt). The log includes a timestamp, and details of the operation performed.

3. **User notification:**

The system uses the JavaMail API to send email notifications for various events: users will receive a confirmation email after successfully registering and completing vehicle rentals.

- **User system: User**

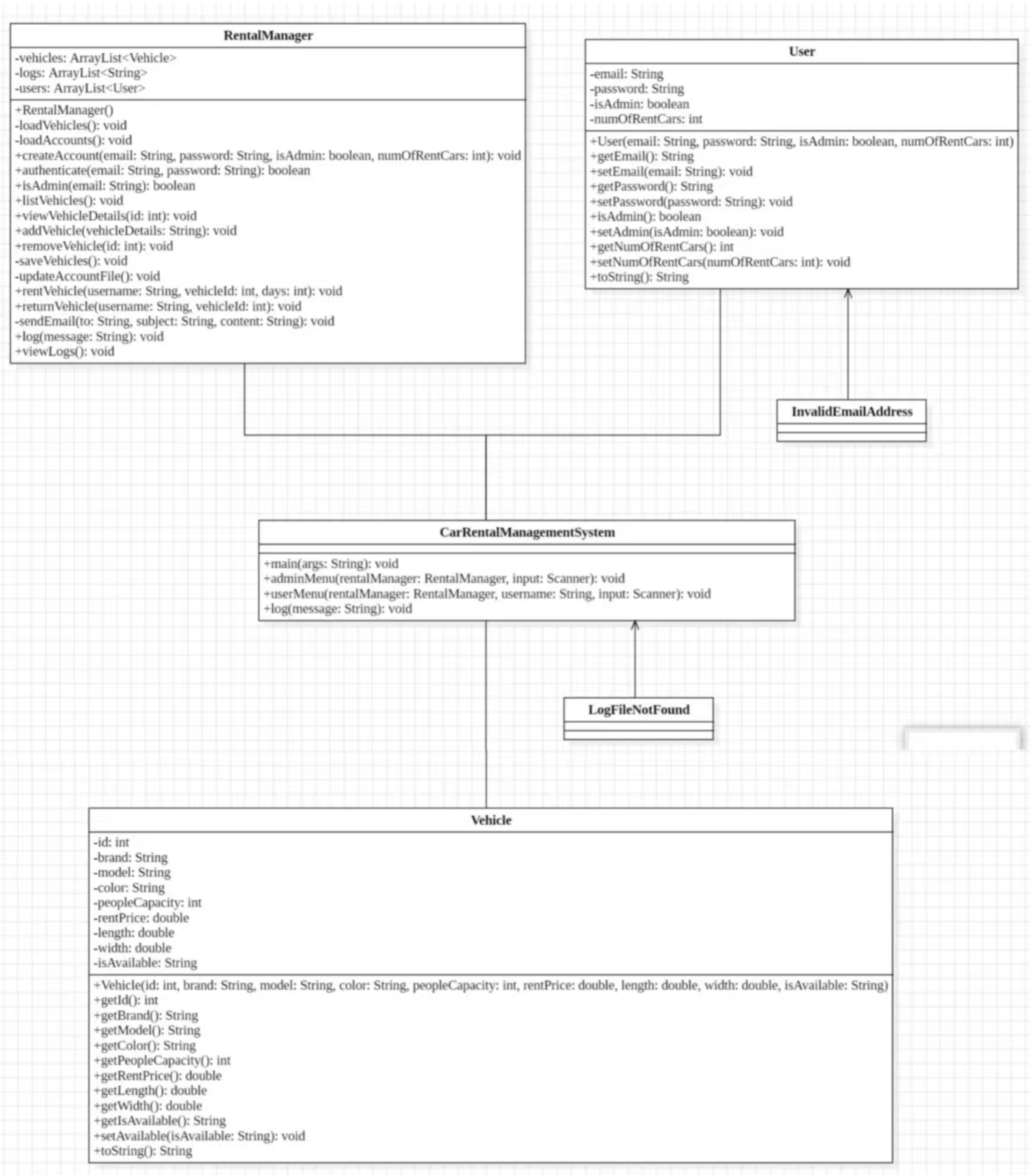
1. **Clear user classification:**

The system clearly distinguishes between ordinary users and administrator users, and different users have different functions, with excellent privacy and classification

2. **Rich function reservation:**

The user system contains the necessary data structures of the menu and user interaction system (CRMS) to ensure that the system can normally implement all the functions of the main class.

(A) UML Diagram



(B) UML Code Generate

CarRentalManagementSystem:

CarRentalManagementSystem.java

```java

import java.io.*;
import java.util.*;

/**
 *
 */
public class CarRentalManagementSystem {

    /**
     * Default constructor
     */
    public CarRentalManagementSystem() {
    }

    /**
     * @param args
     * @return
     */
    public void main(String args) {
        // TODO implement here
        return null;
    }

    /**
     * @param rentalManager
     * @param input
     * @return
     */
    public void adminMenu(RentalManager rentalManager, Scanner input) {
        // TODO implement here
        return null;
    }

    /**
     * @param rentalManager
     * @param username
     * @param input
     * @return
     */

    public void userMenu(RentalManager rentalManager, String username, Scanner input) {
        // TODO implement here
        return null;
    }

    /**
     * @param message
     * @return
     */
    public void log(String message) {
        // TODO implement here
        return null;
    }

}
```

RentalManager:

CarRentalManagementSystem.java | RentalManager.java

```java

import java.io.*;
import java.util.*;

/**
 *
 */
public class RentalManager {

    /**
     * Default constructor
     */
    public RentalManager() {
    }

    /**
     *
     */
    private ArrayList<Vehicle> vehicles;

    /**
     *
     */
    private ArrayList<String> logs;

    /**
     *
     */
    private ArrayList<User> users;

    /**
     *
     */
    public RentalManager() {
        // TODO implement here
    }

    /**
     * @return
     */
    private void loadVehicles() {
        // TODO implement here
```

```
        return null;
    }

    /**
     * @return
     */
    private void loadAccounts() {
        // TODO implement here
        return null;
    }

    /**
     * @param email
     * @param password
     * @param isAdmin
     * @param numOfRentCars
     * @return
     */
    public void createAccount(String email, String password, boolean isAdmin, int numOfRentCars) {
        // TODO implement here
        return null;
    }

    /**
     * @param email
     * @param password
     * @return
     */
    public boolean authenticate(String email, String password) {
        // TODO implement here
        return false;
    }

    /**
     * @param email
     * @return
     */
    public boolean isAdmin(String email) {
        // TODO implement here
        return false;
    }

    /**
     * @return
     */
    public void listVehicles() {
        // TODO implement here
        return null;
    }

    /**
     * @param id
     * @return
     */
    public void viewVehicleDetails(int id) {
        // TODO implement here
        return null;
    }

    /**
     * @param vehicleDetails
     * @return
     */
    public void addVehicle(String vehicleDetails) {
        // TODO implement here
        return null;
    }

    /**
     * @param id
     * @return
     */
    public void removeVehicle(int id) {
        // TODO implement here
        return null;
    }

```

```
    /**
     * @return
     */
    private void saveVehicles() {
        // TODO implement here
        return null;
    }

    /**
     * @return
     */
    private void updateAccountFile() {
        // TODO implement here
        return null;
    }

    /**
     * @param username
     * @param vehicleId
     * @param days
     * @return
     */
    public void rentVehicle(String username, int vehicleId, int days) {
        // TODO implement here
        return null;
    }

    /**
     * @param username
     * @param vehicleId
     * @return
     */
    public void returnVehicle(String username, int vehicleId) {
        // TODO implement here
        return null;
    }

    /**
     * @param to
     * @param subject
     * @param content
     * @return
     */
    private void sendEmail(String to, String subject, String content) {
        // TODO implement here
        return null;
    }

    /**
     * @param message
     * @return
     */
    public void log(String message) {
        // TODO implement here
        return null;
    }

    /**
     * @return
     */
    public void viewLogs() {
        // TODO implement here
        return null;
    }

}
```

User:

CarRentalManagementSystem.java | RentalManager.java | User.java

```java

import java.io.*;
import java.util.*;

/**
 *
 */
public class User {

    /**
     * Default constructor
     */
    public User() {
    }

    /**
     *
     */
    private String email;

    /**
     *
     */
    private String password;

    /**
     *
     */
    private boolean isAdmin;

    /**
     *
     */
    private int numOfRentCars;

    /**
     * @param email
     * @param password
     * @param isAdmin
     * @param numOfRentCars
     */

    public User(String email, String password, boolean isAdmin, int numOfRentCars) {
        // TODO implement here
    }

    /**
     * @return
     */
    public String getEmail() {
        // TODO implement here
        return "";
    }

    /**
     * @param email
     * @return
     */
    public void setEmail(String email) {
        // TODO implement here
        return null;
    }

    /**
     * @return
     */
    public String getPassword() {
        // TODO implement here
        return "";
    }

    /**
     * @param password
     * @return
     */
    public void setPassword(String password) {
        // TODO implement here
        return null;
    }

    /**
```

```
     * @return
     */
    public boolean isAdmin() {
        // TODO implement here
        return false;
    }

    /**
     * @param isAdmin
     * @return
     */
    public void setAdmin(boolean isAdmin) {
        // TODO implement here
        return null;
    }

    /**
     * @return
     */
    public int getNumOfRentCars() {
        // TODO implement here
        return 0;
    }

    /**
     * @param numOfRentCars
     * @return
     */
    public void setNumOfRentCars(int numOfRentCars) {
        // TODO implement here
        return null;
    }

    /**
     * @return
     */
```

```
    public String toString() {
        // TODO implement here
        return "";
    }

}
```

Vehicle:

CarRentalManagementSystem.java | RentalManager.java | User.java | Vehicle.java

```java

import java.io.*;
import java.util.*;

/**
 *
 */
public class Vehicle {

    /**
     * Default constructor
     */
    public Vehicle() {
    }

    /**
     *
     */
    private int id;

    /**
     *
     */
    private String brand;

    /**
     *
     */
    private String model;

    /**
     *
     */
    private String color;

    /**
     *
     */
    private int peopleCapacity;

    /**
     *
     */
    private double rentPrice;

    /**
     *
     */
    private double length;

    /**
     *
     */
    private double width;

    /**
     *
     */
    private String isAvailable;

    /**
     * @param id
     * @param brand
     * @param model
     * @param color
     * @param peopleCapacity
     * @param rentPrice
     * @param length
     * @param width
     * @param isAvailable
     */
    public Vehicle(int id, String brand, String model, String color, int peopleCapacity, double rentPrice, double length, double width, String isAvailable) {
        // TODO implement here
    }

    /**
     * @return
     */
    public int getId() {
        // TODO implement here
        return 0;
    }
```

```java

    /**
     * @return
     */
    public String getBrand() {
        // TODO implement here
        return "";
    }

    /**
     * @return
     */
    public String getModel() {
        // TODO implement here
        return "";
    }

    /**
     * @return
     */
    public String getColor() {
        // TODO implement here
        return "";
    }

    /**
     * @return
     */
    public int getPeopleCapacity() {
        // TODO implement here
        return 0;
    }

    /**
     * @return
     */
    public double getRentPrice() {
        // TODO implement here
        return 0.0d;
    }

    /**
     * @return
     */
    public double getLength() {
        // TODO implement here
        return 0.0d;
    }

    /**
     * @return
     */
    public double getWidth() {
        // TODO implement here
        return 0.0d;
    }

    /**
     * @return
     */
    public String getIsAvailable() {
        // TODO implement here
        return "";
    }

    /**
     * @param isAvailable
     * @return
     */
    public void setAvailable(String isAvailable) {
        // TODO implement here
        return null;
    }

    /**
     * @return
     */
    public String toString() {
        // TODO implement here
            return "";
        }

}
```

(C) Full Code Implantation (& Screenshots)

This project has 4 classes: CarRentalManagementSystem, RentalManager, User, and Vehicle. Among them, CarRentalManagementSystem is the startup class.

1. CarRentalManagementSystem

This startup class introduces BufferedWriter, FileWriter, and IOException in the java.io class for reading and modifying desktop data files and collecting error information; it also introduces the java.util.Scanner class to read the commands entered by the user on the console; and finally introduces the java.util.Date class to mark the time of the Log file content.

The main method source code is as follows:

CarRentalManagementSystem.java × RentalManager.java Us

```
1  package Project;
2
3  import java.io.BufferedWriter;
4  import java.io.FileWriter;
5  import java.io.IOException;
6  import java.util.Scanner;
7  import java.util.Date;
8
```

```java
public class CarRentalManagementSystem {
    public static void main(String[] args) {
        RentalManager rentalManager = new RentalManager();
        Scanner input = new Scanner(System.in);

        boolean authenticated = false;
        String username = null;
        boolean isAdmin = false;

        System.out.println("Welcome to the Car Rental Management System!");

        while (!authenticated) {
            System.out.print("Enter your email (or \"new\" to create a new account): ");
            username = input.nextLine();

            if (username.equals("new")) {
                System.out.print("Enter your email: ");
                String newEmail = input.nextLine();
                System.out.print("Enter your password: ");
                String newPassword = input.nextLine();
                System.out.print("What's your role? (user or admin, if you are admin, type \"admin\" exactly; " +
                        "otherwise you will be user): ");
                String role = input.nextLine();
                isAdmin = role.equalsIgnoreCase( anotherString: "admin");

                if (isAdmin) {
                    System.out.print("Enter the admin verification password: ");
                    String adminVerificationPassword = input.nextLine();
                    if (!adminVerificationPassword.equals("Ethan12345")) {
                        System.out.println("Invalid admin verification password. Account creation failed.");
                        continue;
                    }
                }

                rentalManager.createAccount(newEmail, newPassword, isAdmin,  numOfRentCars: 0);
                log("Created new account for " + newEmail);
                System.out.println("Account created! Please log in.");
            } else {
                System.out.print("Enter your password: ");
                String password = input.nextLine();

                isAdmin = rentalManager.isAdmin(username);

                if (rentalManager.authenticate(username, password)) {
                    authenticated = true;
                    log(username + " successfully logged in.");
                    System.out.println("Welcome, " + username + "!");
                } else {
                    System.out.println("Invalid email or password. Please try again.");
                }
            }
        }

        while (true) {
            if (isAdmin) {
                adminMenu(rentalManager, input);
            } else {
                userMenu(rentalManager, username, input);
            }
        }
    }
```

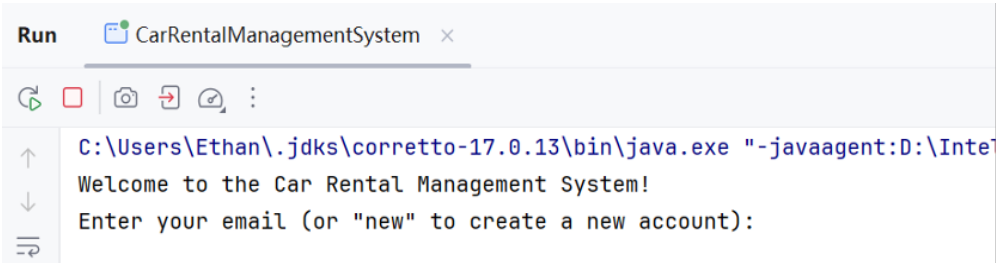
```
public static void adminMenu(RentalManager rentalManager, Scanner input) {  1 usage
    System.out.println("\nAdmin Menu:");
    System.out.println("1. Add a vehicle");
    System.out.println("2. Remove a vehicle");
    System.out.println("3. View all vehicles");
    System.out.println("4. View rental logs");
    System.out.println("5. Exit");

    System.out.print("Enter your choice: ");
    int choice = input.nextInt();
    input.nextLine();

    switch (choice) {
        case 1:
            System.out.print("Enter vehicle details (format: Brand,Model,Color,Capacity,RentPrice,Length,Width): ");
            String vehicleDetails = input.nextLine();
            rentalManager.addVehicle(vehicleDetails);
            log("Admin added a new vehicle: " + vehicleDetails);
            break;
        case 2:
            System.out.print("Enter vehicle ID to remove: ");
            int vehicleId = input.nextInt();
            rentalManager.removeVehicle(vehicleId);
            log("Admin removed vehicle ID: " + vehicleId);
            break;
        case 3:
            rentalManager.listVehicles();
            break;
        case 4:
            rentalManager.viewLogs();
            break;
        case 5:
            System.out.println("Goodbye!");
            System.exit( status: 0);
        default:
            System.out.println("Invalid choice. Please try again.");
    }
}
```

```
public static void userMenu(RentalManager rentalManager, String username, Scanner input) {  1 usage
    System.out.println("\nUser Menu:");
    System.out.println("1. View all vehicles");
    System.out.println("2. View vehicle details");
    System.out.println("3. Rent a vehicle");
    System.out.println("4. Return a vehicle");
    System.out.println("5. Exit");

    System.out.print("Enter your choice: ");
    int choice = input.nextInt();
    input.nextLine();

    switch (choice) {
        case 1:
            rentalManager.listVehicles();
            break;
        case 2:
            System.out.print("Enter vehicle ID to view details: ");
            int vehicleId = input.nextInt();
            rentalManager.viewVehicleDetails(vehicleId);
            break;
        case 3:
            System.out.print("Enter vehicle ID to rent: ");
            int rentVehicleId = input.nextInt();
            System.out.print("Enter rental period (in days): ");
            int rentalPeriod = input.nextInt();
            rentalManager.rentVehicle(username, rentVehicleId, rentalPeriod);
            break;
        case 4:
            System.out.print("Enter vehicle ID to return: ");
            int returnVehicleId = input.nextInt();
            rentalManager.returnVehicle(username, returnVehicleId);
            break;
        case 5:
            System.out.println("Goodbye!");
            System.exit( status: 0);
        default:
            System.out.println("Invalid choice. Please try again.");
    }
}
```

```
public static void log(String message) {  4 usages
    try (BufferedWriter writer = new BufferedWriter(new FileWriter( fileName: "C:\\Users\\Ethan\\Desktop\\Log.txt",  append: true))) {
        writer.write( str: new Date() + ": " + message);
        writer.newLine();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
```

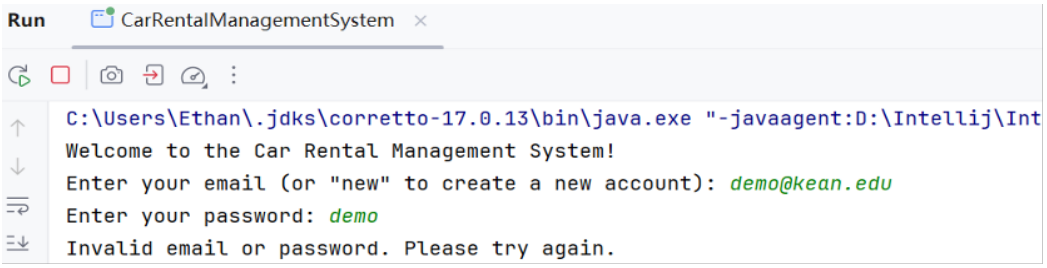
The main method will use the RentalManager class to first create a rentalManager object, and then guide the user to choose whether to log in or create a new account.

The **demonstration** is as follows:

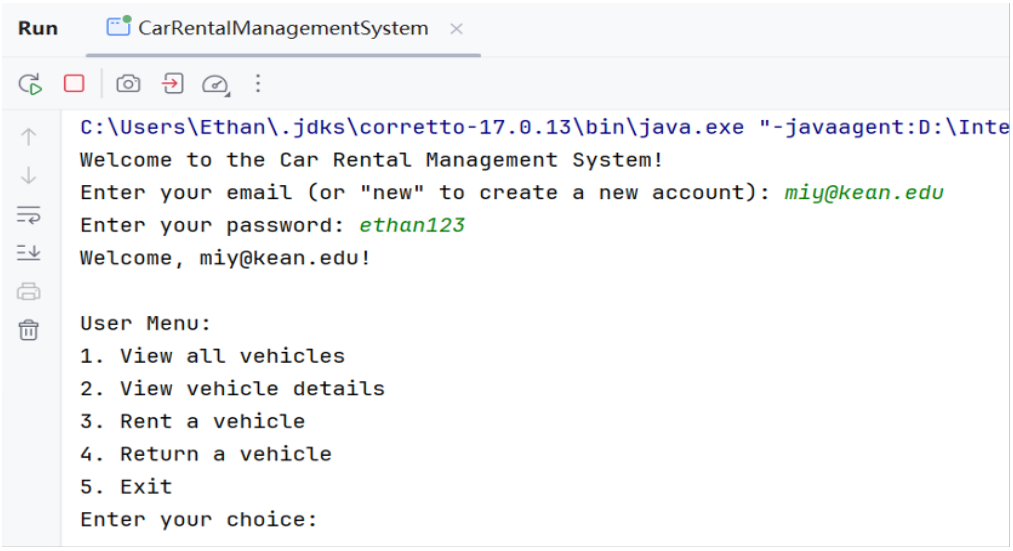


If you choose to log in, the system will read the existing account from the Accounts.txt file on the desktop through java.io.

If the account does not exist or the password is wrong, the system will prompt the user to re-enter it

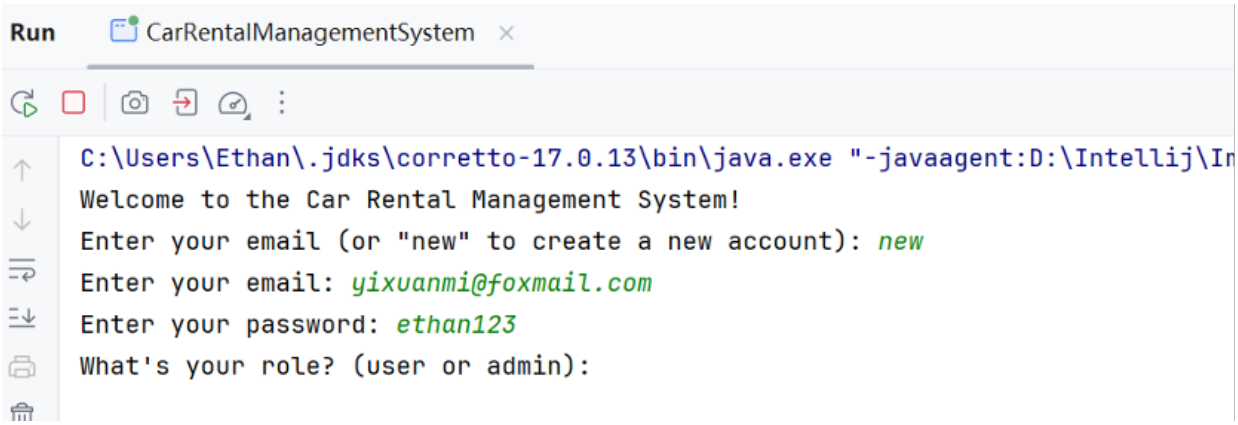


If the account exists and the password is correct, the system menu will be successfully logged in, prompting the user to enter the operation command



If you choose to create a new user, the system will prompt you to enter your account and password.

Then you will be asked whether to create an administrator user or a normal user.



If you create a normal user, it will be created successfully directly; if you create an administrator user, you need to enter a confirmation password (the default in the project is Ethan12345)

```
Enter your email (or "new" to create a new account): new
Enter your email: ethan@cmu.edu
Enter your password: ethan123
What's your role? (user or admin): admin
Enter the admin verification password: Ethan12345
Account created! Please log in.
```

The **user mode** code is shown below:

```
109 @    public static void userMenu(RentalManager rentalManager, String username, Scanner input) {  1 usage
110          System.out.println("\nUser Menu:");
111          System.out.println("1. View all vehicles");
112          System.out.println("2. View vehicle details");
113          System.out.println("3. Rent a vehicle");
114          System.out.println("4. Return a vehicle");
115          System.out.println("5. Exit");
116
117          System.out.print("Enter your choice: ");
118          int choice = input.nextInt();
119          input.nextLine();
120
121          switch (choice) {
122              case 1:
123                  rentalManager.listVehicles();
124                  break;
125              case 2:
126                  System.out.print("Enter vehicle ID to view details: ");
127                  int vehicleId = input.nextInt();
128                  rentalManager.viewVehicleDetails(vehicleId);
129                  break;
130              case 3:
131                  System.out.print("Enter vehicle ID to rent: ");
132                  int rentVehicleId = input.nextInt();
133                  System.out.print("Enter rental period (in days): ");
134                  int rentalPeriod = input.nextInt();
135                  rentalManager.rentVehicle(username, rentVehicleId, rentalPeriod);
136                  break;
137              case 4:
138                  System.out.print("Enter vehicle ID to return: ");
139                  int returnVehicleId = input.nextInt();
140                  rentalManager.returnVehicle(username, returnVehicleId);
141                  break;
142              case 5:
143                  System.out.println("Goodbye!");
144                  System.exit( status: 0);
```

```
145              default:
146                  System.out.println("Invalid choice. Please try again.");
147          }
148      }
149
```

In **user mode**, there are four operations in the menu.

The first is to view the rough information of all vehicles. Here, the Vehicles.txt file on the desktop is read through io. The **demonstration** is as follows:

```
4. Return a vehicle
5. Exit
Enter your choice: 1
```

ID	Brand	Model	Color	Capacity	RentPrice	Length	Width
1	Toyota	Corolla	White	5	50.00	4.50	1.80
2	Honda	Civic	Black	5	60.00	4.70	1.90
3	Ford	Focus	Blue	5	55.00	4.60	1.80
4	Nissan	Altima	Gray	5	70.00	4.90	1.90
5	Chevrolet	Malibu	Red	5	65.00	4.80	1.90
6	BMW	3 Series	White	5	100.00	4.70	1.80
7	Mercedes-Benz	C-Class	Black	5	120.00	4.80	1.90
8	Audi	A4	Silver	5	110.00	4.70	1.90
9	Volkswagen	Passat	Blue	5	75.00	4.90	1.90
10	Hyundai	Elantra	Gray	5	50.00	4.50	1.80
11	Tesla	Model 3	White	5	130.00	4.70	1.80
12	Tesla	Model Y	Black	7	150.00	4.80	2.00
13	Toyota	RAV4	Silver	5	80.00	4.60	1.90
14	Ford	Escape	Blue	5	85.00	4.60	1.90
15	Jeep	Wrangler	Red	5	90.00	4.70	2.10
16	Kia	Sorento	Gray	7	70.00	4.80	1.90
17	Hyundai	Tucson	White	5	60.00	4.50	1.80
18	Honda	CR-V	Black	5	75.00	4.60	1.90
19	Mazda	CX-5	Blue	5	70.00	4.50	1.80
20	Subaru	Outback	Green	5	75.00	4.80	1.90
21	Volvo	XC60	Silver	5	100.00	4.70	1.90
22	Lexus	RX 350	Gray	5	110.00	4.70	1.90
23	Chevrolet	Traverse	White	8	90.00	5.00	2.00
24	Nissan	Rogue	Black	5	70.00	4.60	1.90
25	BMW	X5	Blue	5	150.00	4.90	2.00
26	Audi	Q5	Silver	5	130.00	4.80	1.90
27	Mercedes-Benz	GLC	Black	5	140.00	4.80	1.90
28	Volkswagen	Tiguan	Blue	7	75.00	4.70	1.90
29	Toyota	Highlander	Red	8	120.00	4.90	2.00
30	Honda	Pilot	Gray	8	110.00	4.90	2.00

Next, view the detailed information of a specific vehicle by vehicle ID, as shown below:

```
User Menu:
1. View all vehicles
2. View vehicle details
3. Rent a vehicle
4. Return a vehicle
5. Exit
Enter your choice: 2
Enter vehicle ID to view details: 21
ID: 21, Brand: Volvo, Model: XC60, Color: Silver, Capacity: 5, Rent Price: $100.0/day, Dimensions: 4.7m x 1.9m, Available: true
```

Then rent a car by vehicle ID. After entering the ID, the system will prompt the user to enter the rental period, and then send a notification email to the specified email address, as shown below:

```
User Menu:
1. View all vehicles
2. View vehicle details
3. Rent a vehicle
4. Return a vehicle
5. Exit
Enter your choice: 3
Enter vehicle ID to rent: 21
Enter rental period (in days): 10



Email sent successfully to miy@kean.edu
Vehicle rented successfully!
```

Then you can return the car by vehicle ID. If the user enters the wrong vehicle ID, it will show that the ID is invalid or the car has not been rented out; if the user has not rented a car but still returns the car, it will show that you are currently renting no car; if the vehicle ID is entered correctly, the car will be returned successfully. The demonstration is as follows:

```
User Menu:
1. View all vehicles
2. View vehicle details
3. Rent a vehicle
4. Return a vehicle
5. Exit
Enter your choice: 4
Enter vehicle ID to return: 22
Invalid vehicle ID or the vehicle was not rented!

User Menu:
1. View all vehicles
2. View vehicle details
3. Rent a vehicle
4. Return a vehicle
5. Exit
Enter your choice: 4
Enter vehicle ID to return: 21
Vehicle returned successfully!

User Menu:
1. View all vehicles
2. View vehicle details
3. Rent a vehicle
4. Return a vehicle
5. Exit
Enter your choice: 4
Enter vehicle ID to return: 21
You have not rented any vehicles!
```

Finally, exit the system, as shown below:

```
User Menu:
1. View all vehicles
2. View vehicle details
3. Rent a vehicle
4. Return a vehicle
5. Exit
Enter your choice: 5
Goodbye!

Process finished with exit code 0
```

The **administrator mode** code is shown below:

```
70 @    public static void adminMenu(RentalManager rentalManager, Scanner input) {  1 usage
71          System.out.println("\nAdmin Menu:");
72          System.out.println("1. Add a vehicle");
73          System.out.println("2. Remove a vehicle");
74          System.out.println("3. View all vehicles");
75          System.out.println("4. View rental logs");
76          System.out.println("5. Exit");
77
78          System.out.print("Enter your choice: ");
79          int choice = input.nextInt();
80          input.nextLine();
81
82          switch (choice) {
83              case 1:
84                  System.out.print("Enter vehicle details (format: Brand,Model,Color,Capacity,RentPrice,Length,Width): ");
85                  String vehicleDetails = input.nextLine();
86                  rentalManager.addVehicle(vehicleDetails);
87                  log("Admin added a new vehicle: " + vehicleDetails);
88                  break;
89              case 2:
90                  System.out.print("Enter vehicle ID to remove: ");
91                  int vehicleId = input.nextInt();
92                  rentalManager.removeVehicle(vehicleId);
93                  log("Admin removed vehicle ID: " + vehicleId);
94                  break;
95              case 3:
96                  rentalManager.listVehicles();
97                  break;
98              case 4:
99                  rentalManager.viewLogs();
100                 break;
101             case 5:
102                 System.out.println("Goodbye!");
103                 System.exit( status: 0);
104             default:
105                 System.out.println("Invalid choice. Please try again.");
106         }
107     }
108
```

In **administrator mode**, you can add vehicles, as shown below:

```
Admin Menu:
1. Add a vehicle
2. Remove a vehicle
3. View all vehicles
4. View rental logs
5. Exit
Enter your choice: 1
Enter vehicle details (format: Brand,Model,Color,Capacity,RentPrice,Length,Width): Porsche,911,Red,2,1500,4.5,2.5
Vehicle added successfully!
```

Check all vehicles and find that there is one more vehicle (30 becomes 31). The demonstration is as follows:

```
| 26   | Audi          | Q5          | Silver   | 5        | 130.00    | 4.80
| 27   | Mercedes-Benz | GLC         | Black    | 5        | 140.00    | 4.80
| 28   | Volkswagen    | Tiguan      | Blue     | 7        | 75.00     | 4.70
| 29   | Toyota        | Highlander  | Red      | 8        | 120.00    | 4.90
| 30   | Honda         | Pilot       | Gray     | 8        | 110.00    | 4.90
| 31   | Porsche       | 911         | white    | 2        | 500.00    | 4.50
--------------------------------------------------------------------------------

Admin Menu:
1. Add a vehicle
```

If the user enters the wrong format of adding a vehicle, it will show that the

```
Admin Menu:
1. Add a vehicle
2. Remove a vehicle
3. View all vehicles
4. View rental logs
5. Exit
Enter your choice: 1
Enter vehicle details (format: Brand,Model,Color,Capacity,RentPrice,Length,Width): Porsche
Invalid vehicle details!
```

The **administrator mode** can also delete the vehicle, as shown below:

```
Admin Menu:
1. Add a vehicle
2. Remove a vehicle
3. View all vehicles
4. View rental logs
5. Exit
Enter your choice: 2
Enter vehicle ID to remove: 31
Vehicle removed successfully!
```

Check all vehicles and find that there are only 30 vehicles and the 31 is deleted.

```
| 25   | BMW           | X5          | Blue     | 5        | 150.00    | 4.90    | 2.00    |
| 26   | Audi          | Q5          | Silver   | 5        | 130.00    | 4.80    | 1.90    |
| 27   | Mercedes-Benz | GLC         | Black    | 5        | 140.00    | 4.80    | 1.90    |
| 28   | Volkswagen    | Tiguan      | Blue     | 7        | 75.00     | 4.70    | 1.90    |
| 29   | Toyota        | Highlander  | Red      | 8        | 120.00    | 4.90    | 2.00    |
| 30   | Honda         | Pilot       | Gray     | 8        | 110.00    | 4.90    | 2.00    |
-----------------------------------------------------------------------------------------
```

The **administrator mode** can view the log file, as shown below:

```
Enter your choice: 4
---- Rental Logs ----
Wed Dec 04 21:22:37 CST 2024: miy@kean.edu successfully logged in.
Wed Dec 04 21:25:09 CST 2024: miy@kean.edu successfully logged in.
Wed Dec 04 21:27:16 CST 2024: Created new account for 1306031@wku.edu.cn
Wed Dec 04 21:27:20 CST 2024: 1306031@wku.edu.cn successfully logged in.
Wed Dec 04 21:30:48 CST 2024: miy@kean.edu successfully logged in.
Wed Dec 04 23:12:40 CST 2024: miy@kean.edu successfully logged in.
Wed Dec 04 23:14:03 CST 2024: Created new account for 1307943@wku.edu.cn
Wed Dec 04 23:14:16 CST 2024: 1307943@wku.edu.cn successfully logged in.
Wed Dec 04 23:21:10 CST 2024: yixuanmi25@gmail.com successfully logged in.
Wed Dec 04 23:24:40 CST 2024: miy@kean.edu successfully logged in.
Thu Dec 05 09:34:41 CST 2024: miy@kean.edu successfully logged in.
Fri Dec 06 00:12:52 CST 2024: miy@kean.edu successfully logged in.
Fri Dec 06 00:34:49 CST 2024: Created new account for yixuanmi@foxmail.com
Fri Dec 06 00:36:31 CST 2024: Created new account for Ethan@cmu.edu
Fri Dec 06 00:37:57 CST 2024: Created new account for ethan@cmu.edu
Fri Dec 06 00:38:29 CST 2024: miy@kean.edu successfully logged in.
Fri Dec 06 00:50:03 CST 2024: miy@kean.edu successfully logged in.
Fri Dec 06 00:55:18 CST 2024: yixuanmi25@gmail.com successfully logged in.
Fri Dec 06 00:55:50 CST 2024: 1307943@wku.edu.cn successfully logged in.
Fri Dec 06 00:56:22 CST 2024: Admin added a new vehicle:
Fri Dec 06 00:56:40 CST 2024: 1307943@wku.edu.cn successfully logged in.
Fri Dec 06 00:57:54 CST 2024: Admin added a new vehicle: Porsche,911,white,2,500,4.5,2
Fri Dec 06 00:58:28 CST 2024: Admin removed vehicle ID: 31
Fri Dec 06 00:58:42 CST 2024: 1307943@wku.edu.cn successfully logged in.
Fri Dec 06 00:59:05 CST 2024: 1307943@wku.edu.cn successfully logged in.
Fri Dec 06 00:59:23 CST 2024: Admin added a new vehicle: Porsche,911,white,2,500,4.5,2
Fri Dec 06 01:01:09 CST 2024: Admin removed vehicle ID: 31
----------------------
```

Finally, exit the system, as shown below:

```
Admin Menu:
1. Add a vehicle
2. Remove a vehicle
3. View all vehicles
4. View rental logs
5. Exit
Enter your choice: 5
Goodbye!
```

If you input an invalid instruction in **user or administrator mode**, both will ask to enter again.

```
User Menu:
1. View all vehicles
2. View vehicle details
3. Rent a vehicle
4. Return a vehicle
5. Exit
Enter your choice: 6
Invalid choice. Please try again.

Admin Menu:
1. Add a vehicle
2. Remove a vehicle
3. View all vehicles
4. View rental logs
5. Exit
Enter your choice: 6
Invalid choice. Please try again.
```

2. RentalManager

The main function of this class is to build the data structure and construction method of the lease, and introduce the javax.mail package to implement the email sending function. The data structure is mainly based on ArrayList. The source code is as follows:

RentalManager.java × User.java Vehicle.java

```java
package Project;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Date;
import javax.mail.*;
import javax.mail.internet.*;
import javax.mail.Session;
import javax.mail.Transport;
import java.util.Properties;

public class RentalManager {  4 usages
    private ArrayList<Vehicle> vehicles;  10 usages
    private ArrayList<String> logs;  2 usages
    private ArrayList<User> users;  8 usages

    public RentalManager() {  1 usage
        vehicles = new ArrayList<>();
        logs = new ArrayList<>();
        users = new ArrayList<>();
        loadVehicles();
        loadAccounts();
    }
```

loadVehicles() will read the vehicle information from the Vehicles.txt file on the desktop

```java
// Load vehicles information from file
private void loadVehicles() {  1 usage
    try (BufferedReader reader = new BufferedReader(new FileReader( fileName: "C:\\Users\\Ethan\\Desktop\\Vehicles.txt"))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] parts = line.split( regex: ",");
            if (parts.length == 9) {
                int id = Integer.parseInt(parts[0].trim());
                String brand = parts[1].trim();
                String model = parts[2].trim();
                String color = parts[3].trim();
                int capacity = Integer.parseInt(parts[4].trim());
                double price = Double.parseDouble(parts[5].trim());
                double length = Double.parseDouble(parts[6].trim());
                double width = Double.parseDouble(parts[7].trim());
                String isAvailable = parts[8].trim();
                vehicles.add(new Vehicle(id, brand, model, color, capacity, price, length, width, isAvailable));
            }
        }
    } catch (IOException e) {
        System.out.println("Error loading vehicles: " + e.getMessage());
    }
}
```

loadAccounts() will load the existing account from the Account.txt file on the desktop and read the password, whether the account is an administrator, and other information from it for subsequent operations

```
54      // Load accounts information from file
55      private void loadAccounts() {  1 usage
56          try (BufferedReader reader = new BufferedReader(new FileReader( fileName: "C:\\Users\\Ethan\\Desktop\\Accounts.txt"))) {
57              String line;
58              while ((line = reader.readLine()) != null) {
59                  String[] parts = line.split( regex: ",");
60                  if (parts.length == 4) {
61                      String email = parts[0].trim();
62                      String password = parts[1].trim();
63                      boolean isAdmin = parts[2].trim().equalsIgnoreCase( anotherString: "admin");
64                      int numOfRentals = Integer.parseInt(parts[3].trim());
65                      users.add(new User(email, password, isAdmin, numOfRentals));
66                  }
67              }
68          } catch (IOException e) {
69              System.out.println("Error loading accounts: " + e.getMessage());
70          }
71      }
```

Creating a user will add the user to the Accounts.txt file on the desktop

```
73      // Create account
74      public void createAccount(String email, String password, boolean isAdmin, int numOfRentCars) {  1 usage
75          users.add(new User(email, password, isAdmin,  numOfRentCars: 0));
76          try (BufferedWriter writer = new BufferedWriter(new FileWriter( fileName: "C:\\Users\\Ethan\\Desktop\\Accounts.txt",  append: true))) {
77              writer.write( str: email + "," + password + "," + (isAdmin ? "admin" : "user") + "," + numOfRentCars);
78              writer.newLine();
79          } catch (IOException e) {
80              System.out.println("Error saving account: " + e.getMessage());
81          }
82      }
```

There are two ways to verify users. One is to check whether the user's account and password are correct, and the other is to detect whether the user is an administrator.

```
84      // Authenticate user
85      public boolean authenticate(String email, String password) {  1 usage
86          for (User user : users) {
87              if (user.getEmail().equals(email) && user.getPassword().equals(password)) {
88                  return true;
89              }
90          }
91          return false;
92      }
93
94      public boolean isAdmin(String email) {  1 usage
95          for (User user : users) {
96              if (user.getEmail().equals(email)) {
97                  return user.isAdmin();
98              }
99          }
100         return false;
101     }
```

These two methods can view all vehicle basic information or view individual vehicle details, using formatted console printing.

```
103     // List all vehicles
104     public void listVehicles() {  2 usages
105         System.out.println("-----------------------------------------------------------------------------------------------------");
106         System.out.printf("| %-4s | %-14s | %-12s | %-8s | %-9s | %-9s | %-7s | %-7s |\n",
107                 "ID", "Brand", "Model", "Color", "Capacity", "RentPrice", "Length", "Width");
108         System.out.println("-----------------------------------------------------------------------------------------------------");
109
110         for (Vehicle vehicle : vehicles) {
111             System.out.printf("| %-4d | %-14s | %-12s | %-8s | %-9d | %-9.2f | %-7.2f | %-7.2f |\n",
112                     vehicle.getId(), vehicle.getBrand(), vehicle.getModel(), vehicle.getColor(),
113                     vehicle.getPeopleCapacity(), vehicle.getRentPrice(), vehicle.getLength(), vehicle.getWidth());
114         }
115         System.out.println("-----------------------------------------------------------------------------------------------------");
116     }
117
118     // View vehicle details
119     public void viewVehicleDetails(int id) {  1 usage
120         // Iterate through the vehicles list to find the vehicle with the given ID
121         for (Vehicle vehicle : vehicles) {
122             if (vehicle.getId() == id) {
123                 System.out.println(vehicle);
124                 return;
125             }
126         }
127         System.out.println("Vehicle not found!");
128     }
```

These two methods are to add or remove vehicles, which will add or delete all vehicle information from Vehicles.txt.

```
130     // Add vehicle
131     public void addVehicle(String vehicleDetails) {  1 usage
132         String[] parts = vehicleDetails.split( regex: ",");
133         if (parts.length == 7) {
134             int id = vehicles.size() + 1;
135             String brand = parts[0].trim();
136             String model = parts[1].trim();
137             String color = parts[2].trim();
138             int capacity = Integer.parseInt(parts[3].trim());
139             double price = Double.parseDouble(parts[4].trim());
140             double length = Double.parseDouble(parts[5].trim());
141             double width = Double.parseDouble(parts[6].trim());
142             vehicles.add(new Vehicle(id, brand, model, color, capacity, price, length, width,  isAvailable: "true"));
143             saveVehicles();
144             System.out.println("Vehicle added successfully!");
145         } else {
146             System.out.println("Invalid vehicle details!");
147         }
148     }
149
150     // Delete vehicle
151     public void removeVehicle(int id) {  1 usage
152         vehicles.removeIf( Vehicle vehicle -> vehicle.getId() == id);
153         saveVehicles();
154         System.out.println("Vehicle removed successfully!");
155     }
```

These two methods are used to save vehicles or update account files

```
157     // Save vehicles information to file
158     private void saveVehicles() {  4 usages
159         try (BufferedWriter writer = new BufferedWriter(new FileWriter( fileName: "C:\\Users\\Ethan\\Desktop\\Vehicles.txt"))) {
160             for (Vehicle vehicle : vehicles) {
161                 writer.write( str: vehicle.getId() + "," + vehicle.getBrand() + "," + vehicle.getModel() + "," +
162                         vehicle.getColor() + "," + vehicle.getPeopleCapacity() + "," +
163                         vehicle.getRentPrice() + "," + vehicle.getLength() + "," + vehicle.getWidth() + "," + vehicle.getIsAvailable());
164                 writer.newLine();
165             }
166         } catch (IOException e) {
167             System.out.println("Error saving vehicles: " + e.getMessage());
168         }
169     }
170
171     private void updateAccountFile() {  2 usages
172         try (BufferedWriter writer = new BufferedWriter(new FileWriter( fileName: "C:\\Users\\Ethan\\Desktop\\Accounts.txt"))) {
173             for (User user : users) {
174                 writer.write( str: user.getEmail() + "," +
175                         user.getPassword() + "," +
176                         (user.isAdmin() ? "admin" : "user") + "," +
177                         user.getNumOfRentCars());
178                 writer.newLine();
179             }
180         } catch (IOException e) {
181             System.out.println("Error updating accounts file: " + e.getMessage());
182         }
183     }
```

In the car rental method, if the car is rented out, an email will be sent to the user, and the isAvaliable property of the car will be changed to false.

Note that the isAvailable in this project does not use the Boolean type, but the string type. This is because only strings can be saved in txt files. If I use Boolean values, then isAvaliable will only be saved in memory, which does not meet the principle of long-term and continuous availability of our design project.

```
185     // Rent vehicle
186     public void rentVehicle(String username, int vehicleId, int days) {  1 usage
187         for (Vehicle vehicle : vehicles) {
188             if (vehicle.getId() == vehicleId && vehicle.getIsAvailable().equals("true")) {
189                 vehicle.setAvailable("false");
190
191                 for (User user : users) {
192                     if (user.getEmail().equals(username)) {
193                         user.setNumOfRentCars(user.getNumOfRentCars() + 1);
194                         updateAccountFile();
195                         break;
196                     }
197                 }
198
199                 sendEmail(username,  subject: "Vehicle Rental Confirmation",
200                         content: "You have rented the vehicle: " + vehicle.getBrand() + " " +
201                                 vehicle.getModel() + " for " + days + " days." + "\n" +
202                                 "Please return the vehicle on time." + "\n" + "Thank you!");
203                 saveVehicles();
204                 System.out.println("Vehicle rented successfully!");
205                 return;
206             }
207         }
208         System.out.println("Vehicle not available!");
209     }
```

When returning a car, this method will change the isAvaliable of the car back to true in the file. If the user has not rented a car or the vehicle ID is entered incorrectly, different types of error prompts will be displayed.

```java
// Return vehicle
public void returnVehicle(String username, int vehicleId) {  1 usage
    for (Vehicle vehicle : vehicles) {
        for (User user : users) {
            if (user.getEmail().equals(username)) {
                if (user.getNumOfRentCars() == 0) {
                    System.out.println("You have not rented any vehicles!");
                    return;
                }

                if (vehicle.getId() == vehicleId && vehicle.getIsAvailable().equals("false")) {
                    vehicle.setAvailable("true");

                    user.setNumOfRentCars(user.getNumOfRentCars() - 1);
                    updateAccountFile();

                    saveVehicles();
                    System.out.println("Vehicle returned successfully!");
                    return;
                }
            }
        }
    }
    System.out.println("Invalid vehicle ID or the vehicle was not rented!");
}
```

Here, based on the QQ mailbox framework and Maven architecture, an email sender is created, a new Session is created, and a confirmation email is sent according to the user's mailbox.

```java
// Send email
private void sendEmail(String to, String subject, String content) {  1 usage
    // QQ mail configuration
    String from = "2509783307@qq.com";
    String password = "hztfrepktsoveaai"; // This is an application-specific password

    Properties properties = new Properties();
    properties.put("mail.smtp.host", "smtp.qq.com");
    properties.put("mail.smtp.port", "587");
    properties.put("mail.smtp.auth", "true");
    properties.put("mail.smtp.starttls.enable", "true");

    // Create a session
    Session session = Session.getInstance(properties, new Authenticator() {
        @Override  no usages
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(from, password);
        }
    });

    try {
        MimeMessage message = new MimeMessage(session);
        message.setFrom(new InternetAddress(from));
        message.addRecipient(Message.RecipientType.TO, new InternetAddress(to));
        message.setSubject(subject);
        message.setText(content);

        Transport.send(message);
        System.out.println("Email sent successfully to " + to);
    } catch (MessagingException e) {
        e.printStackTrace();
        System.out.println("Failed to send email to " + to);
    }
}
```

The first operation is used to write all operations to the operation log Log.txt, and the second operation is to view the log in the console (only administrators can do this)

```
272     public void log(String message) {  no usages
273         String logMessage = new Date() + ": " + message;
274         logs.add(logMessage);
275 
276         try (BufferedWriter writer = new BufferedWriter(new FileWriter( fileName: "C:\\Users\\Ethan\\Desktop\\Log.txt",  append: true))) {
277             writer.write(logMessage);
278             writer.newLine();
279         } catch (IOException e) {
280             System.out.println("Error writing to log file: " + e.getMessage());
281         }
282     }
283 
284     public void viewLogs() {  1 usage
285         String logFilePath = "C:\\Users\\Ethan\\Desktop\\Log.txt";
286 
287         try (BufferedReader reader = new BufferedReader(new FileReader(logFilePath))) {
288             String line;
289             System.out.println("---- Rental Logs ----");
290             while ((line = reader.readLine()) != null) {
291                 System.out.println(line);
292             }
293             System.out.println("----------------------");
294         } catch (FileNotFoundException e) {
295             System.out.println("Log file not found.");
296         } catch (IOException e) {
297             System.out.println("Error reading log file: " + e.getMessage());
298         }
299     }
300 }
```

3. User

The User class serves the system's users or administrators. The attribute used to determine whether the user is an administrator is isAdmin, and it also contains most getter and setter methods. The source code is as follows:

User.java × Vehicle.java

```java
package Project;

public class User {  8 usages
    private String email;  4 usages
    private String password;  3 usages
    private boolean isAdmin;  4 usages
    private int numOfRentCars;  3 usages

    // Constructor
    public User(String email, String password, boolean isAdmin, int numOfRentCars) {  2 usages
        this.email = email;
        this.password = password;
        this.isAdmin = isAdmin;
        this.numOfRentCars = numOfRentCars;
    }

    public String getEmail() {  5 usages
        return email;
    }

    public void setEmail(String email) {  no usages
        this.email = email;
    }

    public String getPassword() {  2 usages
        return password;
    }

    public void setPassword(String password) {  no usages
        this.password = password;
    }

    public boolean isAdmin() {  no usages
        return isAdmin;
    }

    public void setAdmin(boolean isAdmin) {  no usages
        this.isAdmin = isAdmin;
    }

    public int getNumOfRentCars() {  4 usages
        return numOfRentCars;
    }

    public void setNumOfRentCars(int numOfRentCars) {  2 usages
        this.numOfRentCars = numOfRentCars;
    }

    @Override
    public String toString() {
        return "User{" +
                "email='" + email + '\'' +
                ", isAdmin=" + isAdmin +
                '}';
    }
}
```

4. Vehicle

The vehicle class creates various attributes of the vehicle, including brand, model, color, etc. It also contains normal getter and setter methods. The source code is as follows:

Vehicle.java

```java
package Project;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class Vehicle {  8 usages
    private int id;  4 usages
    private String brand;  4 usages
    private String model;  4 usages
    private String color;  4 usages
    private int peopleCapacity;  4 usages
    private double rentPrice;  4 usages
    private double length;  4 usages
    private double width;  4 usages
    private String isAvailable;  4 usages

    public Vehicle(int id, String brand, String model, String color, int peopleCapacity,  2 usages
                   double rentPrice, double length, double width, String isAvailable) {
        this.id = id;
        this.brand = brand;
        this.model = model;
        this.color = color;
        this.peopleCapacity = peopleCapacity;
        this.rentPrice = rentPrice;
        this.length = length;
        this.width = width;
        this.isAvailable = isAvailable; // 默认可用
    }

    // Getters
    public int getId() {  6 usages
        return id;
    }

    public String getBrand() {  3 usages
        return brand;
    }

    public String getModel() {  3 usages
        return model;
    }

    public String getColor() {  2 usages
        return color;
    }

    public int getPeopleCapacity() {  2 usages
        return peopleCapacity;
    }

    public double getRentPrice() {  2 usages
        return rentPrice;
    }

    public double getLength() {  2 usages
        return length;
    }

    public double getWidth() {  2 usages
        return width;
    }

    public String getIsAvailable() {  3 usages
        return isAvailable;
    }

}
```

```
68      // Setters
69  @   public void setAvailable(String isAvailable) {  2 usages
70          this.isAvailable = isAvailable;
71          if (isAvailable.equals("false")) {
72              try {
73                  FileWriter fileWriter = new FileWriter( fileName: "C:\\Users\\Ethan\\Desktop\\Vehicles.txt",  append: true);
74                  BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
75                  bufferedWriter.write( str: "\n" + id + ", " + brand + ", " + model + ", " + color + ", " + peopleCapacity
76                          + ", " + rentPrice + ", " + length + ", " + width + ", " + isAvailable);
77                  bufferedWriter.close();
78              } catch (IOException e) {
79                  e.printStackTrace();
80              }
81          }
82      }
83
84      @Override
85      public String toString() {
86          return "ID: " + id + ", Brand: " + brand + ", Model: " + model +
87                  ", Color: " + color + ", Capacity: " + peopleCapacity +
88                  ", Rent Price: $" + rentPrice + "/day, Dimensions: " +
89                  length + "m x " + width + "m, Available: " + isAvailable;
90      }
91  }
```

IV. Analysis and Report

(a)Discussion

- **Data Structure Selection**

In this project, the ArrayList data structure is used, which is tailored for specific scenarios to optimize system performance and improve system maintainability. The following is an analysis of its space and time complexity and advantages:

ArrayList

Insertion operation (insert at the end): O(1)

Search operation: O(n)

Delete operation: O(n)

Advantages:

Sequential storage and traversal are efficient. The system uses ArrayList to store vehicle information and user accounts, which allows fast access to individual records. Supports random access to elements, making operations such as displaying vehicle details simple.

- **Exception Handling**

Exception handling is an important part of the system to ensure stability and enhance user experience. For example:

1. **Transaction failure:**

If a user tries to rent an unavailable vehicle, a clear error message is displayed to guide the user to retry with valid input; if the user does not rent any vehicle, the system prevents the user from returning the vehicle and displays a clear error message.

2. **Email failure:**

The system uses an error handling mechanism to solve email sending problems, ensure retry attempts and log failures for administrator review. Failed to enter account and password: If the

user's account and password are entered incorrectly, a clear error message will be displayed.

- **Optimization of actual scenarios**

1. **Input validation:**

To prevent invalid input, a strong validation mechanism is implemented. These include format checks (e.g., ensuring email correctness) and range checks (e.g., valid rental period)

2. **Efficient processing of big data:**

The project is completely based on the ArrayList data structure, which improves read and write speeds and minimizes storage overhead.

- **Email function integration**

The email function is an important function of the system that enables real-time communication: When the user completes the car rental, he receives a confirmation email for the successful transaction.

- **Car rental system management**

The system integrates a management terminal that enables system administrators to perform key operations such as adding, deleting, and updating vehicle records. The terminal also provides access to user information and transaction logs, thereby improving management efficiency and maintaining operational transparency.

- **Summary and Outlook**

Through these data structures and implementing functions optimized for actual scenarios, the system achieves stability, efficiency, and user-friendliness. However, as a project that is constantly being improved, our future optimization of the project will focus on expanding functionality, improving user experience based on feedback, and integrating emerging technologies to meet changing business needs, ensuring that the system remains adaptable and able to respond to new challenges and opportunities.

V. Conclusions

Developed for efficient car rental business, the car rental management system successfully addresses challenges in resource management, user interaction, and management oversight. The system provides a comprehensive solution for administrators and general users, streamlining the car rental process and improving operational efficiency.

By implementing the system, we expect to achieve the following key achievements:

- **Proper resource management:**

The system enables companies to efficiently manage their fleets by maintaining a comprehensive database of vehicle information, including availability status, specifications, and rental history. This ensures that vehicles are effectively utilized while providing administrators with detailed operational insights.

- **Enhanced user experience:**

For customers, the system provides a user-friendly interface that simplifies the car rental process. Users can easily register for an account, browse available vehicles based on various attributes, and complete rental transactions with minimal effort. The integration of email notifications further enhances the experience by providing timely updates and confirmations. Using a private data structure and setting password verification, the security of users is well guaranteed with the security of the system.

- **Strong administrative control:**

Administrators are equipped with powerful tools to add, update, and delete vehicles, as well as monitor user transactions and rental logs. These features not only increase operational transparency, but also help in quick response to any issues or queries. We open different levels of permissions for administrators and users.

- **Efficient data processing:**

From a technical perspective, the system solves challenges such as efficient data storage, fast retrieval, and seamless user interaction. By adopting ArrayList, the system implements different functions, including loading vehicle information, user authentication, etc. In addition, powerful exception handling and input validation mechanisms ensure the stability and reliability of the system.

- **Future Outlook:**

Although there is still room for improvement in the future, such as improving the email notification system and further optimizing the filtering mechanism, the car rental management system represents an important step in using technology to improve resource utilization and user satisfaction. Its successful implementation highlights the importance of innovative system design in addressing real-world challenges and creating value for enterprises and customers.

References

Bajracharya, D., & Kathmandu, N. A REVIEW ON JAVA HASHMAP AND TREEMA

Gil, J., & Shimron, Y. (2011, October). Smaller footprint for java collections. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion (pp. 191-192).

Jacobson, N., & Thornton, A. (2004). It is time to emphasize arraylists over arrays in Java- based first programming courses. In Working group reports from ITiCSE on Innovation and technology in computer science education (pp. 88-92).

Long, F., Mohindra, D., Seacord, R., & Svoboda, D. (2010). Java Concurrency Guidelines. M. Zheng, J. Yang, M. Wen, H. Zhu, Y. Liu and H. Jin, "Why Do Developers Remove Lambda Expressions in Java?," 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021, pp. 67-78, doi10.1109/ASE51524.2021.9678600.

M. Mrena, M. Varga and M. Kvassay, "Experimental Comparison of Array-based and Linked-based List Implementations," 2022 IEEE 16th International Scientific Conference on Informatics (Informatics), Poprad, Slovakia, 2022, pp. 231-238, doi: 10.1109/Informatics57926.2022.10083495.

Osman, M. N., Zain, N. M., Paidi, Z., Sedek, K. A., NajmuddinYusoff, M., & Maghribi, M. (2017). Online car rental system using web-based and SMS technology. *Computing Research & Innovation (CRINN)*, *2*, 277.

Qiu, D., Li, B., & Leung, H. (2016). Understanding the API usage in Java. *Information and software technology*, *73*, 81-100.

Qiu Xingjie. (2001). Urban car rental system. Shanghai Urban Planning, (3), 42-42.

Reges, S., & Stepp, M. (2014). Building Java Programs. Pearson.

Thakur, A. (2021). Car rental system. *Int. J. Res. Appl. Sci. Eng. Technol*, *9*(7), 402-412.

Appendix:

Contribution of Group Members

Student Name: Mi Yixuan Student Number: 1307943

Make UML master table, code integration, project experiment debug, report writing

Student Name: Yu Yiduo Student Number: 1306057

CarRentalManagementSystem class design, report writing

Student Name: Zhao Yiyi Student Number: 1305974

Rental Manager class design, report writing

Student Name: Yu Qiyang Student Number: 1306031

User class design and establishment of relevant user structure, report writing

Student Name: Jia Taoyin Student Number: 1306194

Vehicle class design and establishment of vehicle database, report writing

Dr. Ken Ehimwenma, PhD.