**DOKUZ EYLUL UNIVERSITY ENGINEERING FACULTY**

**DEPARTMENT OF COMPUTER ENGINEERING**

**CME 2210**

**Object Oriented Analysis and Design**

RENTAL CAR MANAGEMENT SYSTEM

**by**

**Hüveyda Başyurtlu Fatma Ceren Akyüz Eray Kubilay**

**30.05.2023**

**CHAPTER ONE**

**INTRODUCTION**

A car rental project is a project that allows users to rent cars online. The purpose of this project is to provide an easy and efficient way for users to search for, reserve, and rent cars from rental company. The program typically involves creating a user-friendly interface, and connecting to files of rental cars and availability.

In this project, users should be able to register and create an account, as well as log in to the system with their email and password. Also, when needed they can manage their accounts, including viewing reservation history and disabling accounts.

To make a reservation for a rental car, they search for available rental cars by location, date, and type of car. And then by selecting a car, entering the rental dates, and providing payment information they complete the car rental process.

Managers should be able to manage the inventory of rental cars, including adding new cars, updating information about existing cars, and removing cars from the system.

The system should check the availability of rental cars and prevent double bookings or overbooking.

**CHAPTER TWO**

**REQUIREMENTS**

**PACKAGES**

**imgpack:** This package includes the images that we use for GUI.

**objectpack:** This package includes the classes that we use for the system. (Car, Customer, Location, Payment, Reservation)

**pagepack:** This package includes the page classes that we use for the run the system and implement GUI operations. (AdminPage, AppPage, CarSelPage, DropOffLocPage, FleetPage, LaunchPage, LocSelPage, Page, PaymentPage, ReservationPage, ResHisPage, SignUpPage)

**CLASSES**

**Customer:** This class represents the user of the car rental site. It contains attributes such as username, id, email, age, phone, password, res, objList. Also, it includes methods such as getters, setters, validation methods, toDocument, writeCSV, readCSV, doCredentialsMatch, findCustomer, deleteAccount.

**Car:** This class represents a rental car. It contains attributes such as brandName, modelName, year, color, fuelConsumption, dailyRentalRate, licencePlate, isReserved, currentLocation. Also, it includes methods such as addToCSV, updateCar, removeCar, getCarList, getters and setters.

**Reservation:** This class represents a reservation made by a user for a rental car. It contains attributes such as the custId, pickupDate, dropOffDate, pickUpTime, dropOffTime, pickupLocation, and dropOffLocation. Also includes methods such as getters and setters, getReservationHistory, addToDocument.

**Payment:** This class represents a payment made by a user for a rental car reservation. It contains attributes such as custId, paymentAmount, paymentDate, paymentTime, CardNumber, ExpirationDate, CVC, and payList. Also, it includes methods such as getters and settes, validations, addToDocument.

**Location:** This class represents a physical location where rental cars can be picked up or dropped off. It contains attributes such as the locationName, address, contactInformation, and isLocationAvailable boolean.

Also it includes methods such as updateLocation, getLocList, getters and setters.

**ATTRIBUTES**

Here are attributes that were included in the classes:

# Customer:

private String userName: holds the name of the user that registered

private String eMail: holds the email of the user that registered

private String password: holds the password of the user that registered

private String idNumber: holds the id number of the user that registered

private int age: holds the age of the user that registered

private String phone: holds the phone of the user that registered

Reservation res: holds reservation of the user that registered

public static List<Customer> objList = new ArrayList<>(): hols the customers

# Car:

private String brandName: holds the brand name of the selected car

private String modelName: holds the model name of the selected car

private String year: holds the year of the selected car

private String color: holds the color of the selected car

private String fuelConsumption: holds the fuel consumption of the selected car

private String dailyRentalRate: holds the daily rental rate of the selected car

private String licensePlate: holds the license plate of the selected car

private boolean isReserved: holds if the car is reserved or not

private String currentLocation: holds the current location of the selected car

# Reservation:

private String custId: holds id of the customer that is using the site

private String pickupDate: holds the pick up date of selected car

private String dropOffDate: holds the drop off date of selected car

private String pickupTime: holds the pick up time of the selected car

private String dropOffTime: holds the drop off time of the selected car

private Location pickupLocation: holds the pick up location of the selected car

private Location dropOffLocation: holds the drop off location of the selected car

Car car: holds the selected car

Payment payment: holds the payment of the customer

public static List <Reservation> resList=new ArrayList<>(): holds the all reservations.

# Payment:

private String custId: holds the id of the customer that make the payment

private String paymentAmount: holds the amount of the payment

private String paymentDate: holds the payment date

private String paymentTime: holds the payment time

private String CardNumber: holds the card number of customer’s card

private String ExpirationDate: holds the expiration date of the customer's card

private String CVC: holds the CVC of the customer's card

public static List <Payment> payList=new ArrayList<>(): holds the all payments.

# Location:

private String locationName: holds the name of the location that selected

private String address: holds the address of the location that selected

private String contactInformation: holds the contact information for selected location

private boolean isLocationAvailable: holds if the selected location is available or not available

**METHODS**

Here are methods that could be included in the classes:

# Customer:

Getters and Setters for attributes needed.

public boolean toDocument(Customer cust) : takes a Customer object and checks if the customer already exists in the Accounts CSV file. If the customer doesn't exist, it adds the customer to the customer list and csv file then returns true.

void writeCSV(List<String> data, String csvFile, String csvSplitBy): adds new customers to the CSV file.

static List<String[]> readCSV(String csvFile, String csvSplitBy) : reads the CSV file and returns a list consisting of arrays containing the attributes of customers.

public static boolean doCredentialsMatch(String eMail, String password): checks whether the email and password information provided by the user attempting to log in are matching or not.

public static Customer findCustomer(String eMail): reads the accounts CSV file and finds the customer that matches the given email.

public void deleteAccount(): deletes the user account if the user requests to delete.

public boolean userNameValidation(Customer cust): checks if the username matches the required pattern

public boolean emailValidation(Customer cust): checks if the email matches the required pattern

public boolean passwordValidation(Customer cust): checks if the password matches the required pattern

public boolean idValidation(Customer cust): checks if the id matches the required pattern

public boolean ageValidation(Customer cust) : checks if the age matches the required pattern

public boolean phoneValidation(Customer cust) : checks if the phone matches the required pattern

boolean allFieldsValidation(Customer cust) : checks if the all fields given above match the required pattern

# Car:

Getters and Setters for attributes needed.

Private void updateCar(double fuelConsumption, double dailyRentalRate, String color): updates the information for an existing rental car in the system

Private void removeCar(String licencePlate): removes a rental car from the system

public void addToCSV(): adds the Car object to the cars csv file

public static void getCarList(DefaultListModel<Car> model, Location l): If the user first selects a location, it adds the suitable cars available in the selected location to the list. If the user first selects a car, it adds the suitable cars to the list .

# Reservation:

Getters and Setters for attributes needed.

public void setPickupDate(): sets the pickup date as instant date on computer

public void setDropOffDate(): sets the drop-off date as 1 day later on computer

public void setPickupTime(): sets the pickup time as instant time on computer

public void setDropOffTime(): sets the drop-off time as instant time on computer

public static void getReservationsHistory(DefaultListModel<String> model, String custId):

displays the reservations that the user has made in the past.

public void addToDocument(Reservation res): adds the reservation information to the CSV

file.

# Payment:

Getters and Setters for attributes needed.

public void setPaymentDate(): sets the payment date as instant date on computer

public void setPaymentTime (): sets the payment time as instant time on computer

public boolean isCardNoValid(Payment pay): checks if the Card No matches the required

pattern

public boolean isCVVValid(Payment pay): checks if the CVV matches the required

pattern

public boolean isExpDateValid(Payment pay): checks if the expiration date matches the

required pattern

boolean allFieldsValidation(Customer cust) : checks if the all fields given above match the

required pattern

public void addToDocument(): adds the payment details to the payment csv file

# Location:

Getters and Setters for attributes needed.

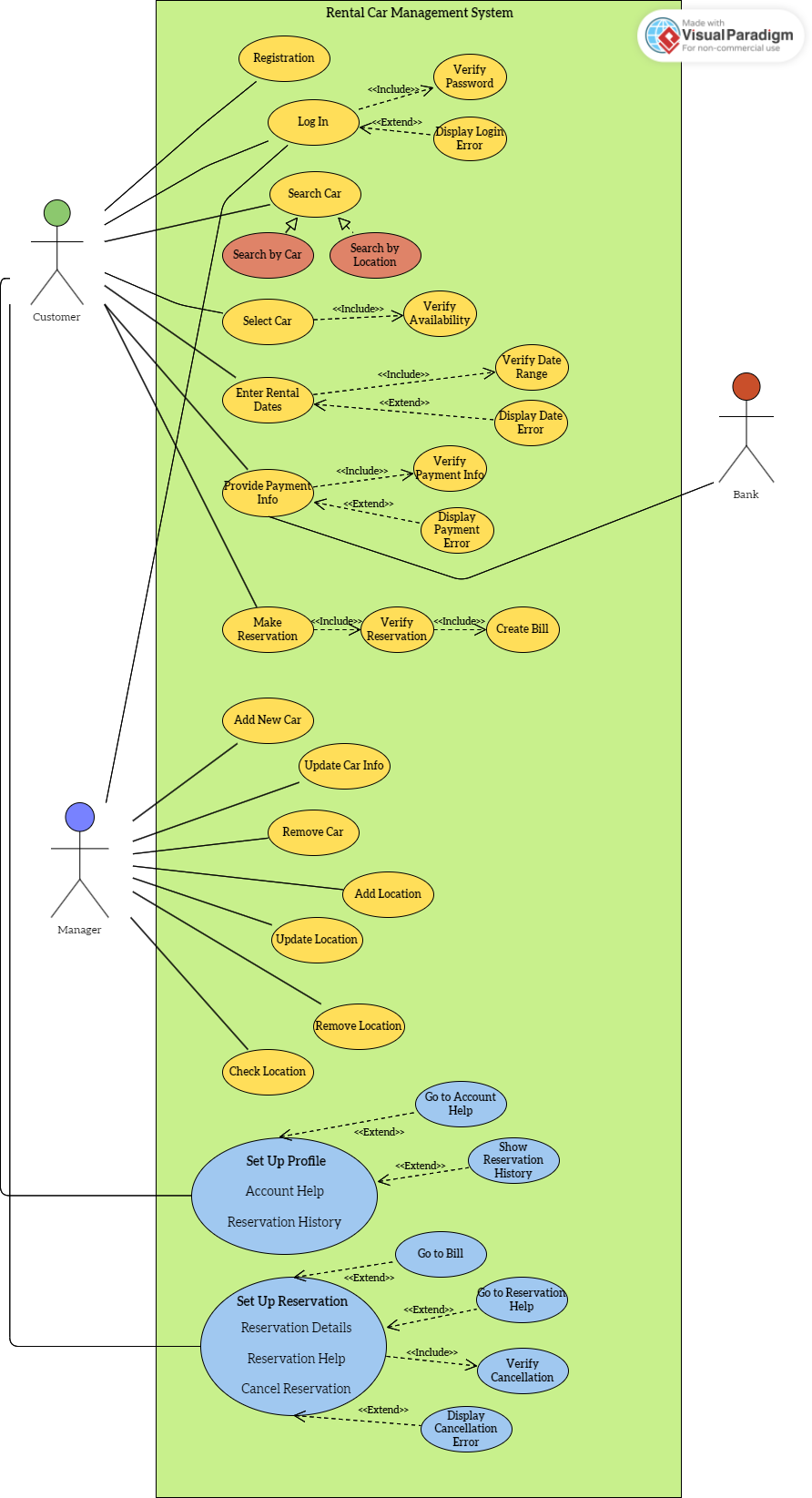
updateLocation(String locationName, String address, Car car): updates the information for an existing location in the system

public static void getLocList(DefaultListModel<Location> model, Car c,int i): If the value of i is 2, it adds the suitable drop-off locations to the list. If the value of i is 1 and if the user has selected a car first it adds the locations where the selected car is available to the list. If the user has selected a location first, it adds the locations where cars are available to the list.

**CHAPTER THREE**

UML DIAGRAMS

**USE CASE DIAGRAM**



The use case diagram shows the system, actors, use cases, and the connections between them.

In our project, the actors are Customer, Bank, and Manager. The customer can create a new account by registering through the system. Afterwards, they log in using the account they created. During login, the correctness of the password must be checked. Therefore, "include" is used. If the user enters the wrong password, a login error message is displayed. Since this situation does not always occur, "extend" is used.

After the customer logs in, they can search for a car either by model or location for rental purposes. They then choose the desired car. The system must check if the selected car is available for rental and whether double booking exists. Therefore, "include" is used. The customer enters the desired rental period. The system must then check if the car is available during the specified rental period. Therefore, "include" is used. If the car is not available during the specified rental period, a time error message is displayed. Since this situation does not always occur, "extend" is used.

Finally, the customer enters their payment information to make a reservation. If the bank confirms the accuracy of the payment information, the system approves the payment. The payment confirmation must be checked. Therefore, "include" is used. If the payment confirmation is not received from the bank, the system displays a payment error message. In this case, "extend" is used. If all of these processes are approved, the system approves the reservation and creates an invoice. Reservation approval and invoice creation are always performed. Therefore, "include" is used.

The manager can add a new car to the system, change the information of an existing car, or delete a car from the system. They can also add a new location to the system, change the information of an existing location, or delete a location from the system. They can also check the reservation status of cars at a location and access car information.

Additionally, the customer may want to modify their profile information, such as changing their information or deleting their account. Therefore, the system directs the customer to the Account help section. Since this situation depends on the customer's request, "extend" is used. If the customer wants to view their reservation history, the system displays the reservation history. This situation also depends on the customer's request. Therefore, "extend" is used. If the customer wants to view their current reservation information, the system displays the invoice information. If the customer wants to make changes to their reservation, the system directs them to the reservation help section. This situation also depends on the customer's request. Therefore, "extend" is used. If the customer wants to cancel their reservation, the system must check if the cancellation request is approved. Therefore, "include" is used. If the cancellation request is not approved, a cancellation error message is displayed. Since this situation does not always occur, "extend" is used.

Diagram

Description automatically generated**CLASS DIAGRAM**

We have 5 classes in our Rental Car Management System. These are Car, Customer, Reservation, Payment, and Location.

Customer class is related to the Car, Payment, and Reservation classes. Relations are:

-Each customer can have 0 or more reservations, but each reservation must belong to only one customer. Also, reservations cannot exist if a customer is deleted.

-Likewise, each customer can have 0 or more payments, but each payment must belong to only one customer. Also, payment cannot exist if a customer is deleted.

-There is a bilateral relationship between car and customer. Each customer can have only one car and each car can only belong to one customer.

Also, the Payment class is related to the Reservation class as follows:

-Each reservation can have only one payment and each payment must belong to only one reservation. Also, if reservation is deleted payment cannot be existing.

There is one more relationship which is between the Car and the Location classes:

-It is a bilateral relationship. Each car must only have one location, but each location can belong to 0 or more cars.

ACTIVITY DIAGRAM



The activity diagram given above shows the interactions that take part in the process of renting the car and doing the payment for it.

As customer presses Rent Car button on the page they get 2 options appearing as buttons, they either pick the location that they would want to pick up a car or they pick the car they want to.

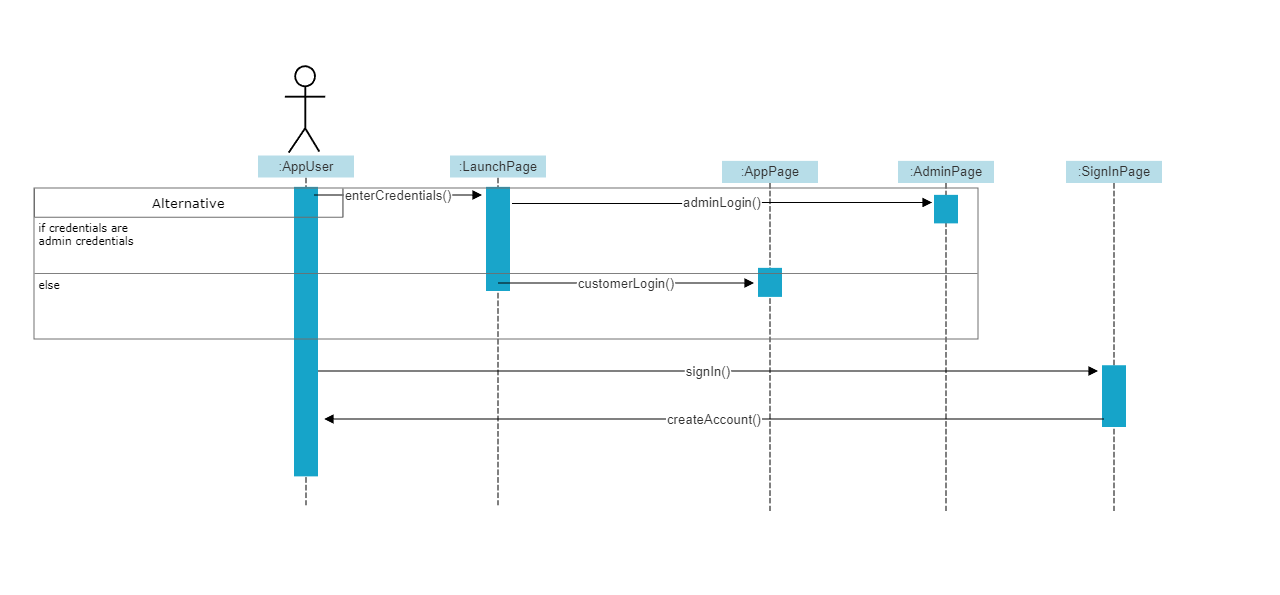
If the customer first chooses the location then they are shown a list of cars that are available on that location to be picked up, they are expected to select the car they want from the list in order to continue.

If the customer chooses the car they want first then they are shown a list of the locations that the chosen car is able to be picked up from, they are expected to select the location they desire from the list in order to continue.

As these actions are completed we expect the customer to enter their credentials for their payment and the credentials should be valid, if they are not we expect the customer to retry.

And if the credentials are valid the payment is successfully done and we receive the payment.

SEQUENCE DIAGRAM

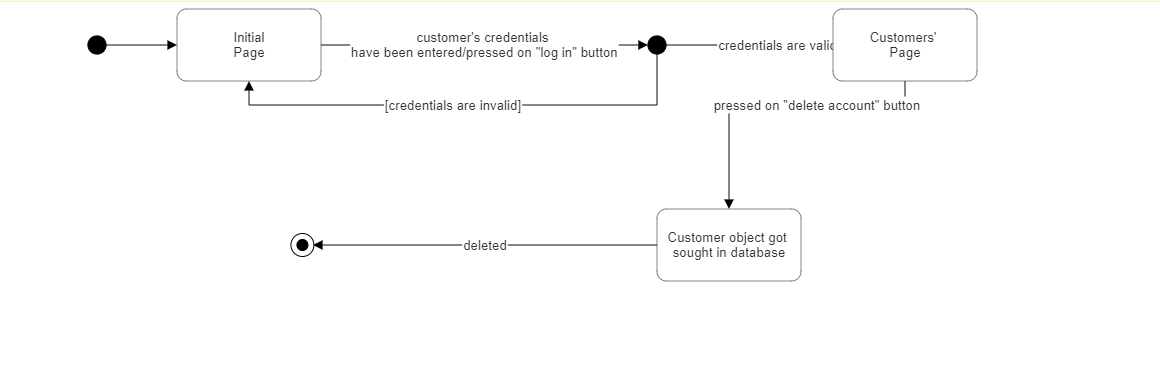


The sequence diagram given above shows the interactions between pages of our app and actor.

As the actor enters their credentials(that exist) on “LaunchPage” they either are admin or customer. If the credentials matches with the admin’s then we launch “AdminPage”. If they don’t then they are customer and we launch “AppPage”.

If the actor doesn’t have an account then they have to press the “Sign In” button in order to create one. This leads us to launch “SignInPage” and expect them to create their account by entering credentials.before they press the “Create Acount” button. As they press on the button we create their account and they are directed back to the “LaunchPage”.

STATE DIAGRAM



The state diagram given above shows the changes in states(pages of our app) as we the customer wants to delete their account.

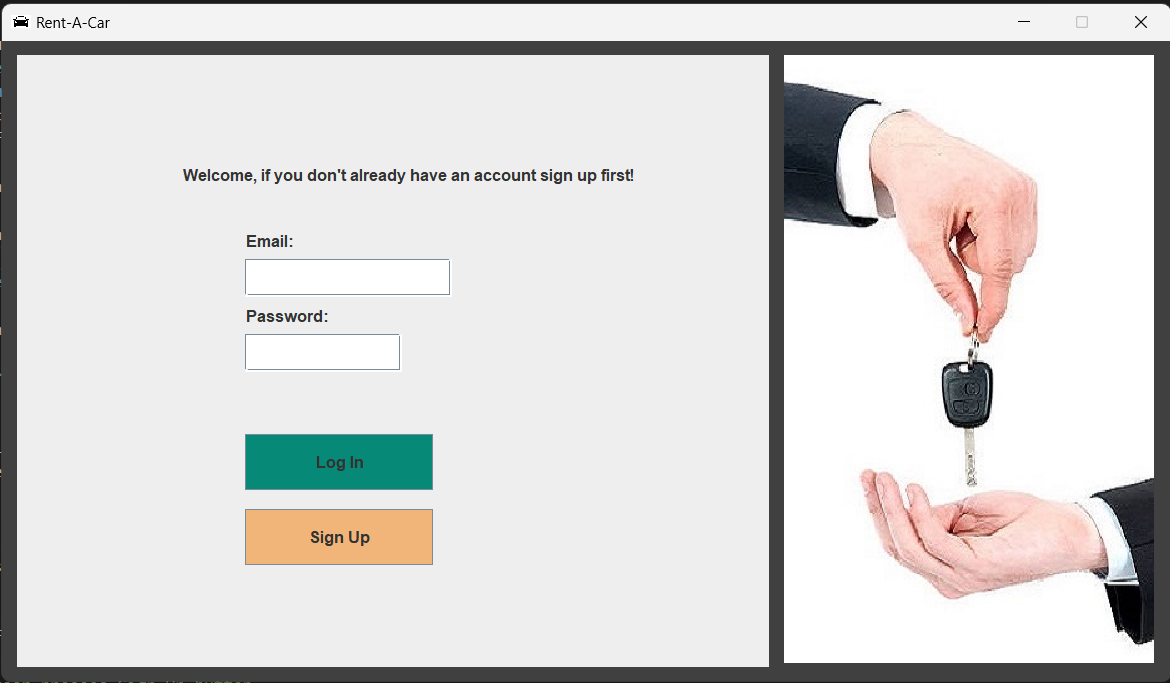
The initial state is our initial page being on the screen, if the customer enters their credentials and presses the “Log In” button we check whether the credentials are valid or invalid. If the credentials are invalid then we return back to the initial state but if they are valid then we change our state as the page changes to “Customers’ Page”. From here we wait for the customer to press on the “Delete Account” button , if this event occurs then we are now in the state where the customer object with the matching credentials got sought, here the deletion action takes part and leads us to the end of this statechart.

# CHAPTER FOUR

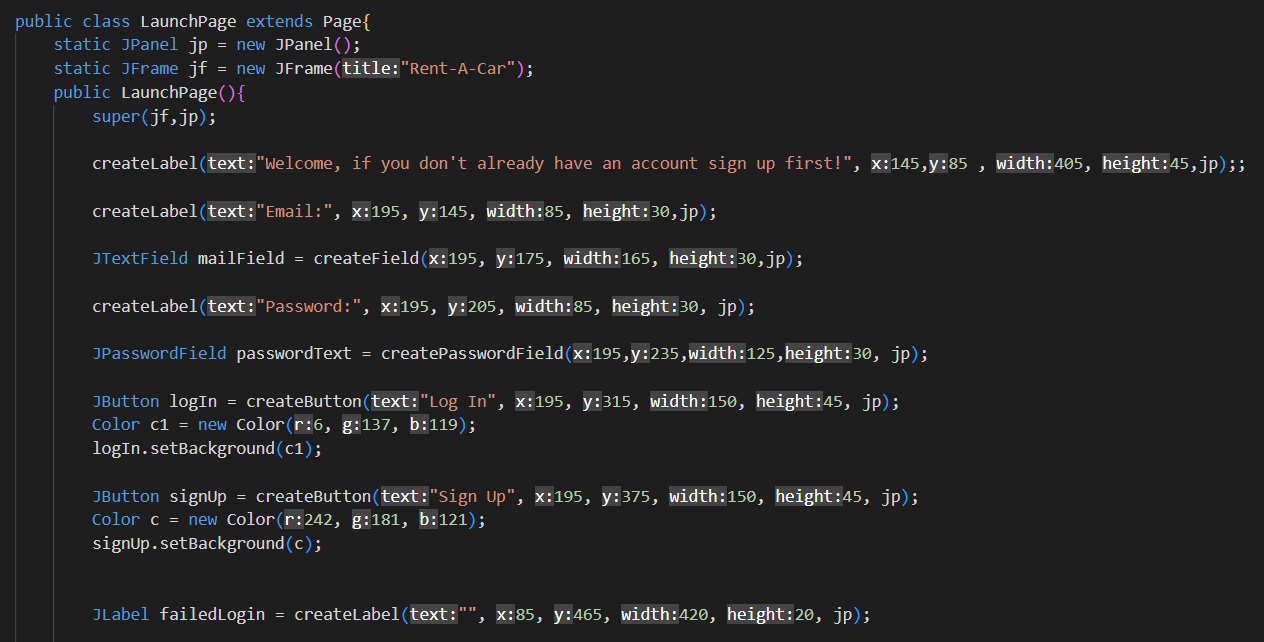
IMPLEMENTATION

We used java swing for the GUI part of the code. We performed the operations by calling the attributes and methods in the Car, Location, Customer, etc. classes required for the application in the page classes we use swing.

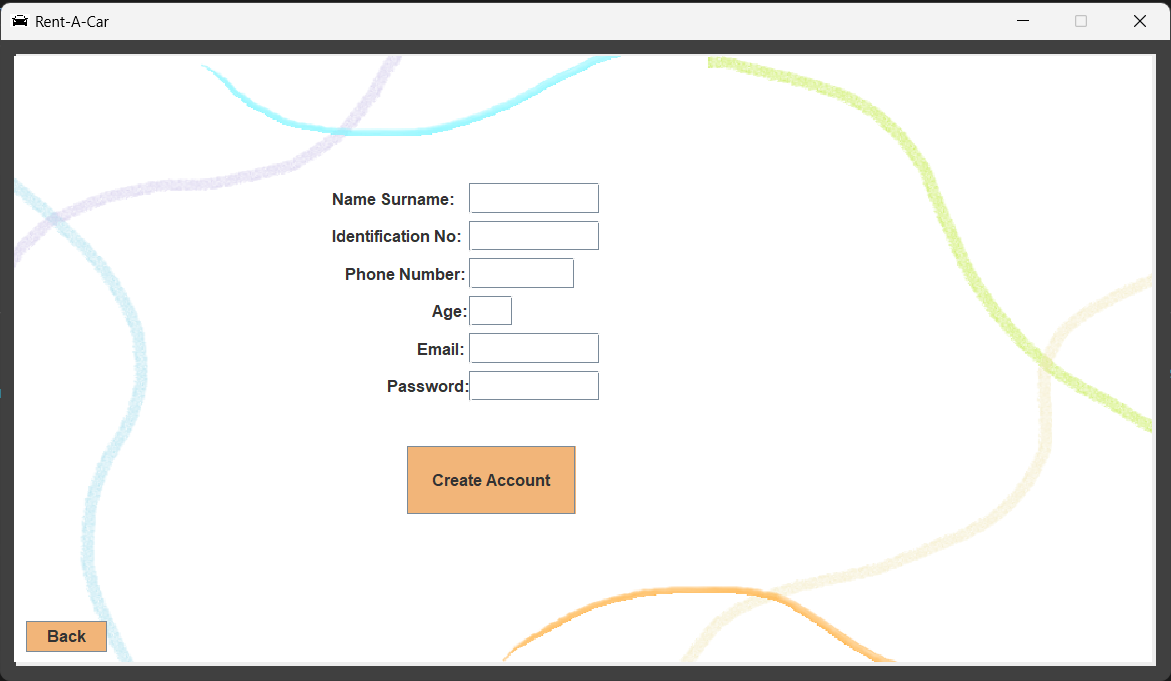
When the code is run for the first time, the login page appears. Launch page does this job.



*(Figure 1. screenshot of launch page)*

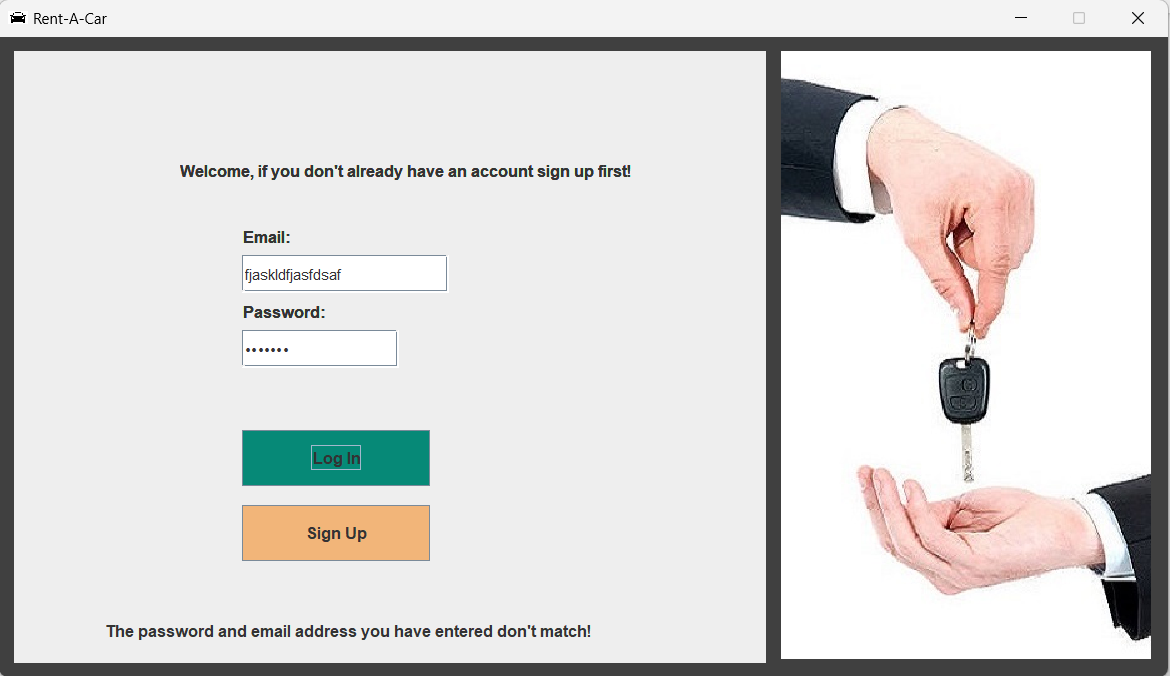
* (Figure 2. screenshot of background code of LauncgPage class)*

Here, if the user is new, he presses the sign up button. And the registration page appears.



*(Figure 3. screenshot of sign up page)*

If the user wants to log in instead of registering and presses the log in button, the entered email and password are checked. If no match is found, it will receive an error message.



*(Figure 4. screenshot of launch page)*

A picture containing text, screenshot, software, multimedia software

Description automatically generated

*(Figure 5. screenshot of background code of LauncgPage class)*

If the user is registering and has entered information suitable for validations, she/he will receive a notification that she has registered and redirected to the launch page to log in.

A screenshot of a computer

Description automatically generated

*(Figure 6. screenshot of sign up page)*

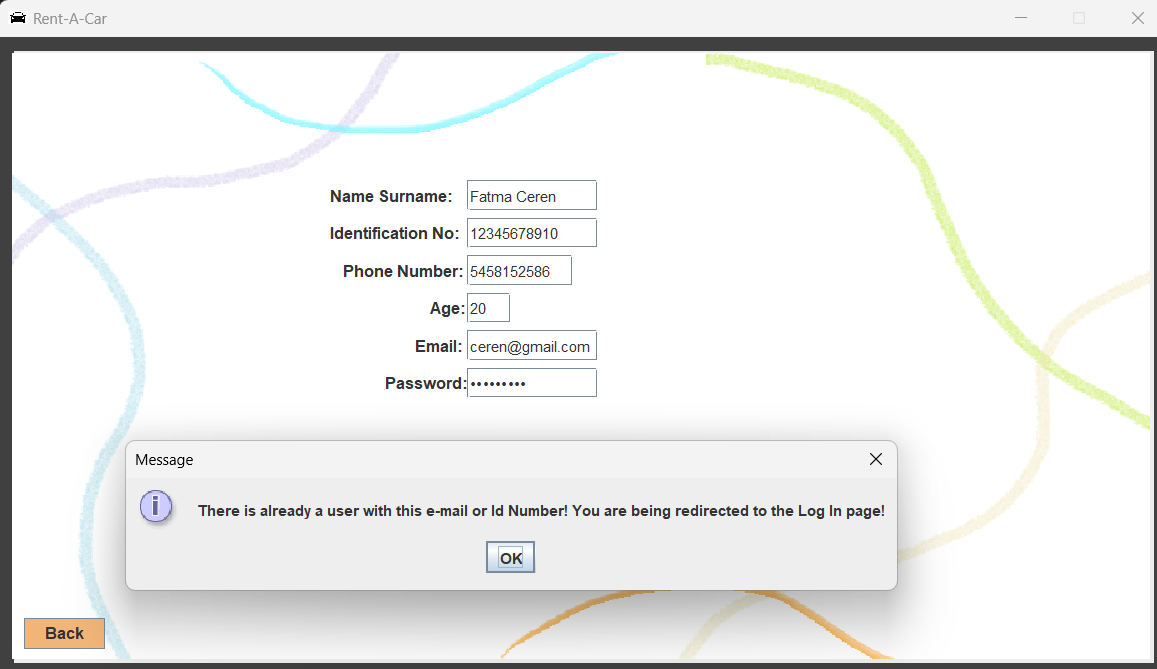
If incorrect information is entered, an error message will be received.

*A screenshot of a computer

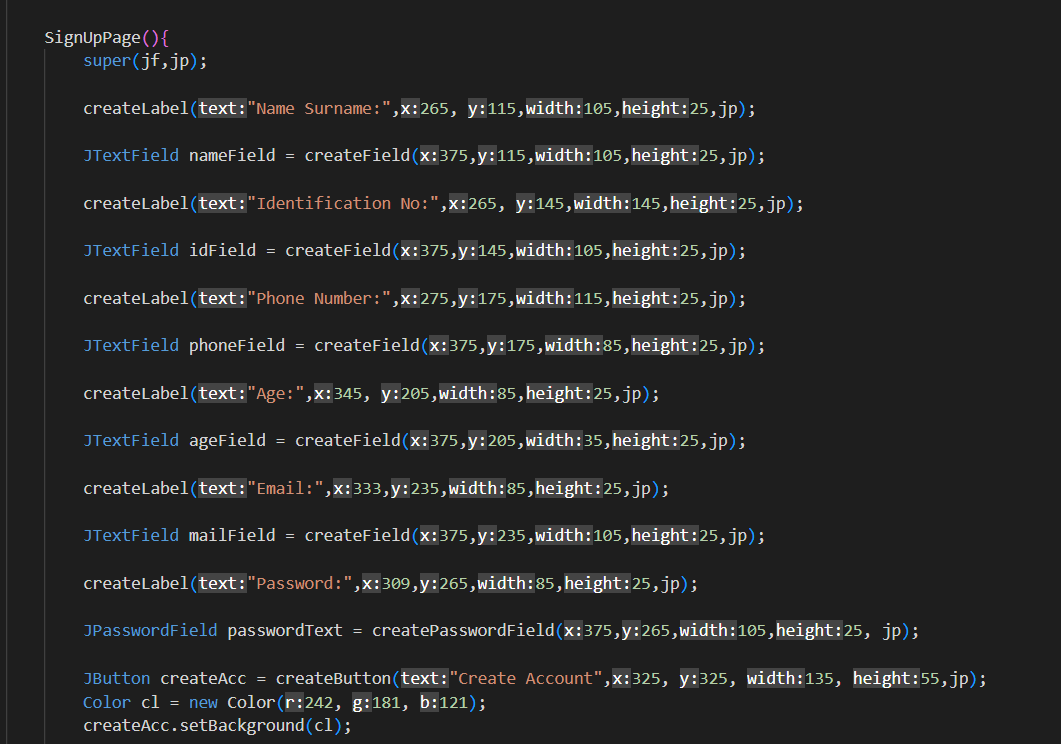
Description automatically generated*

*(Figure 7. screenshot of sign up page)*

If she/he enters a registered email or id, she will receive a warning message and be directed to the launch page.



*(Figure 8. screenshot of sign up page)*

**

*(Figure 9. screenshot of background code of SignUpPage class)*

If the registration is successful, the customer object is created.

*A picture containing text, screenshot, software, computer

Description automatically generated*

*(Figure 10. screenshot of background code of SignUpPage class)*

If the user logs in successfully, she/he is directed to the page where she can choose a car or location.

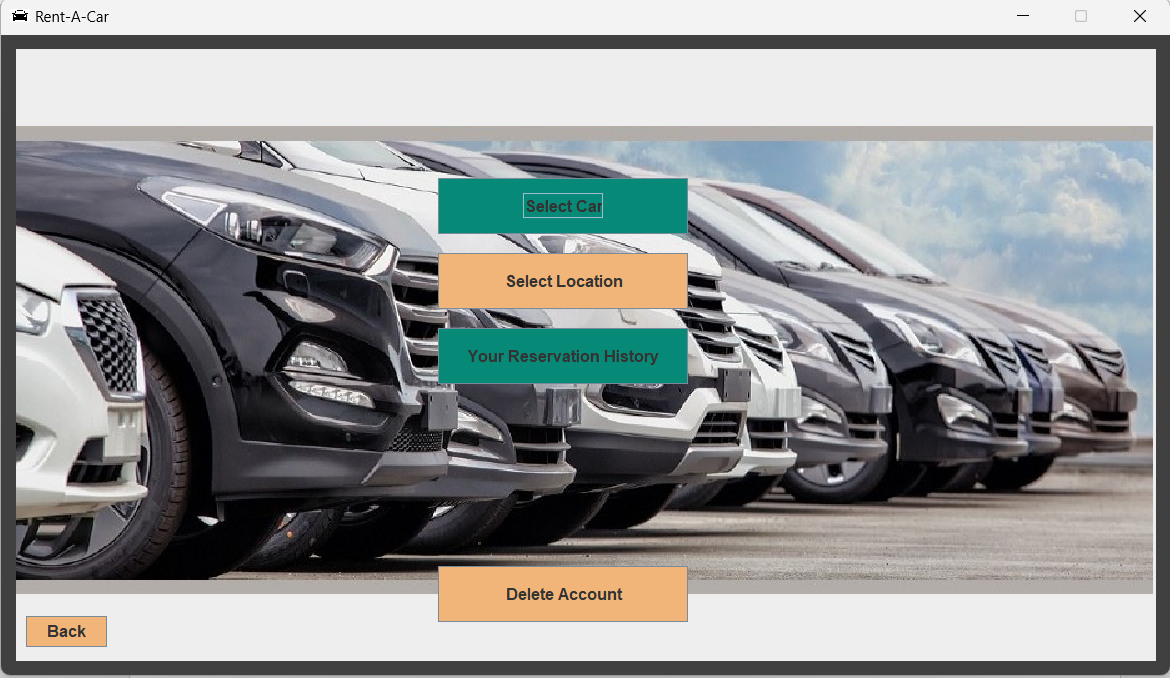
There are two different entry options. Admin and user login. If the user is logged in, he is directed to the app page after entering the email and password. If the admin is logged in, it will be directed to the admin page.

*A screen shot of a computer program

Description automatically generated with low confidence*

*(Figure 10. screenshot of background code of LaunchPage class)*

**If the customer is logged in**, she/he can do four actions. Select car, choose location or view booking history and delete account.



*(Figure 11. screenshot of app page)*

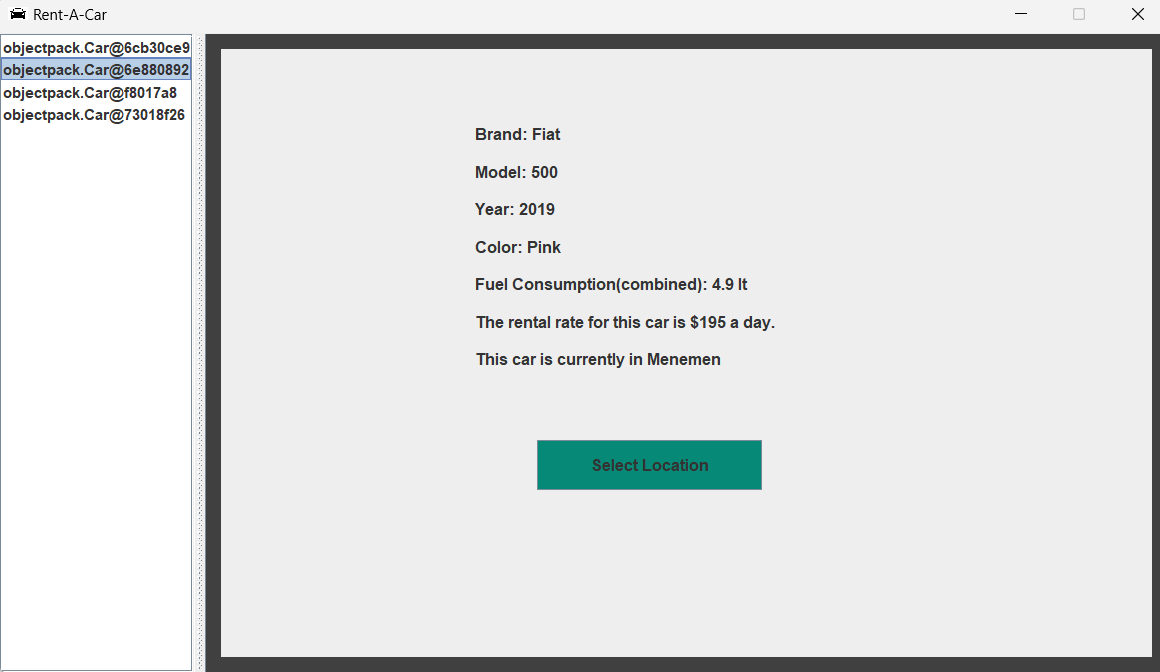
**If the customer clicks on the select car button**, she/he will be directed to the car selection page.

A picture containing text, screenshot, software, multimedia software

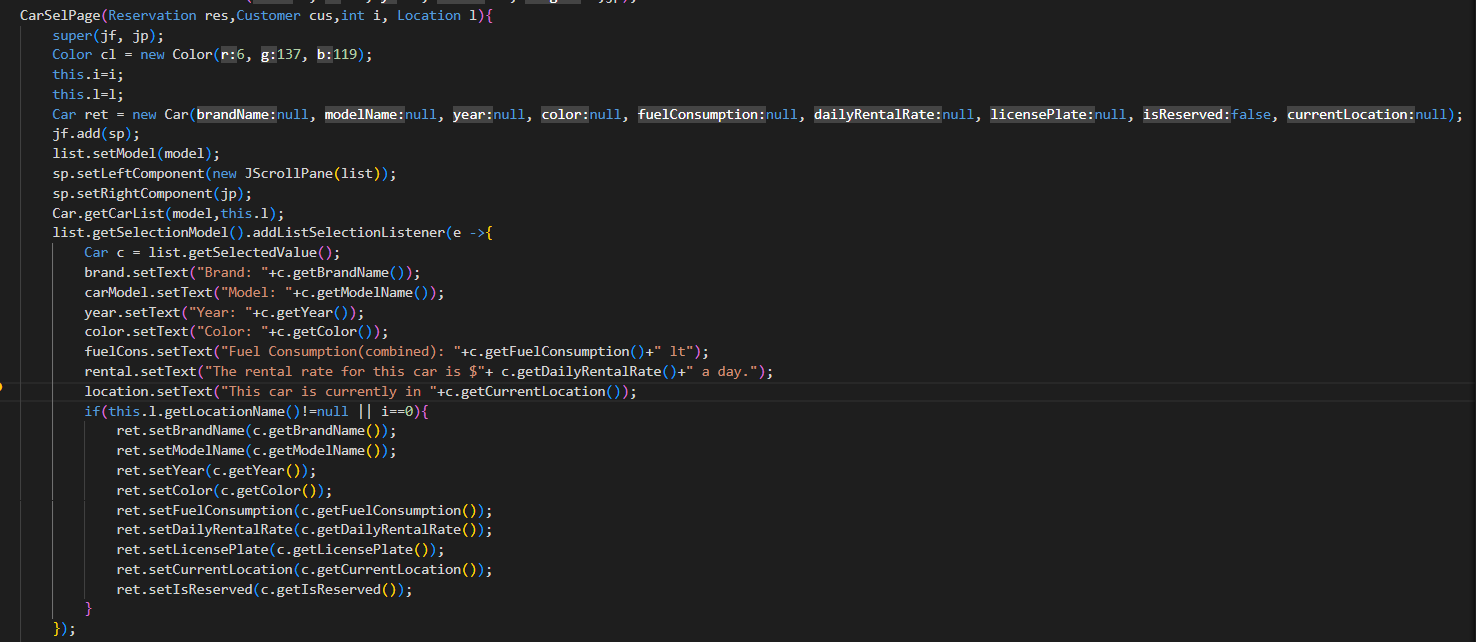
Description automatically generated

*(Figure 12. screenshot of background code of appPage class)*

She/He sees the information of the car she/he has chosen on the car page and can press the select location button to see the locations of the car she/he has chosen.

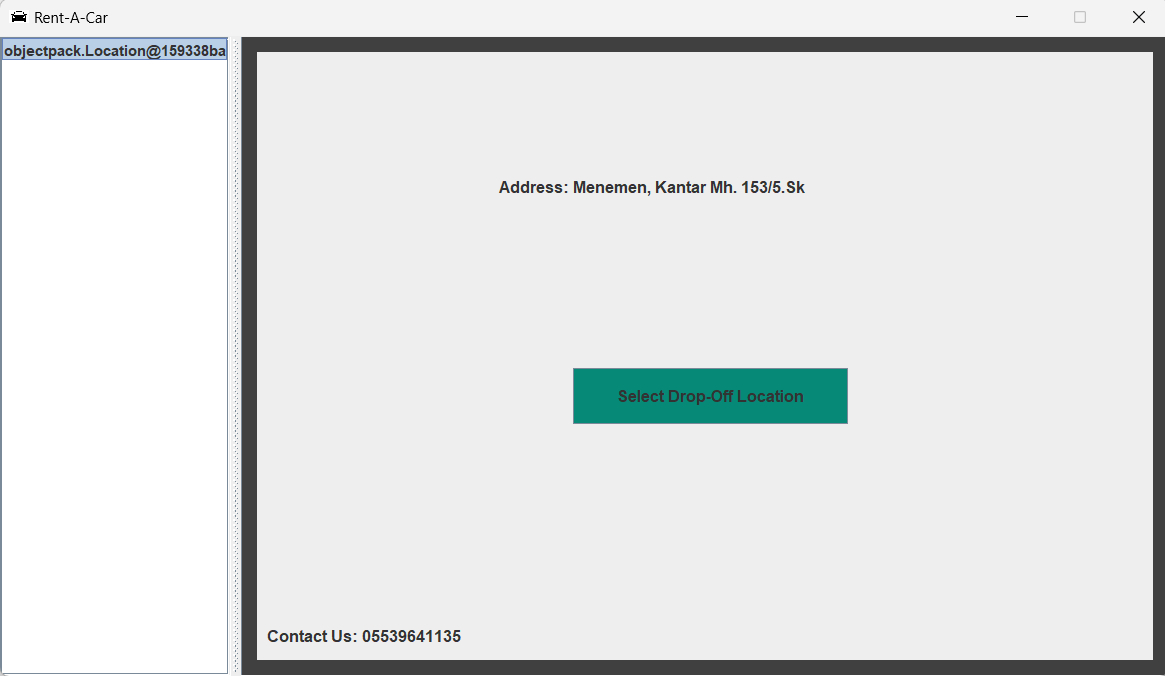


*(Figure 13. screenshot of car selection page)*

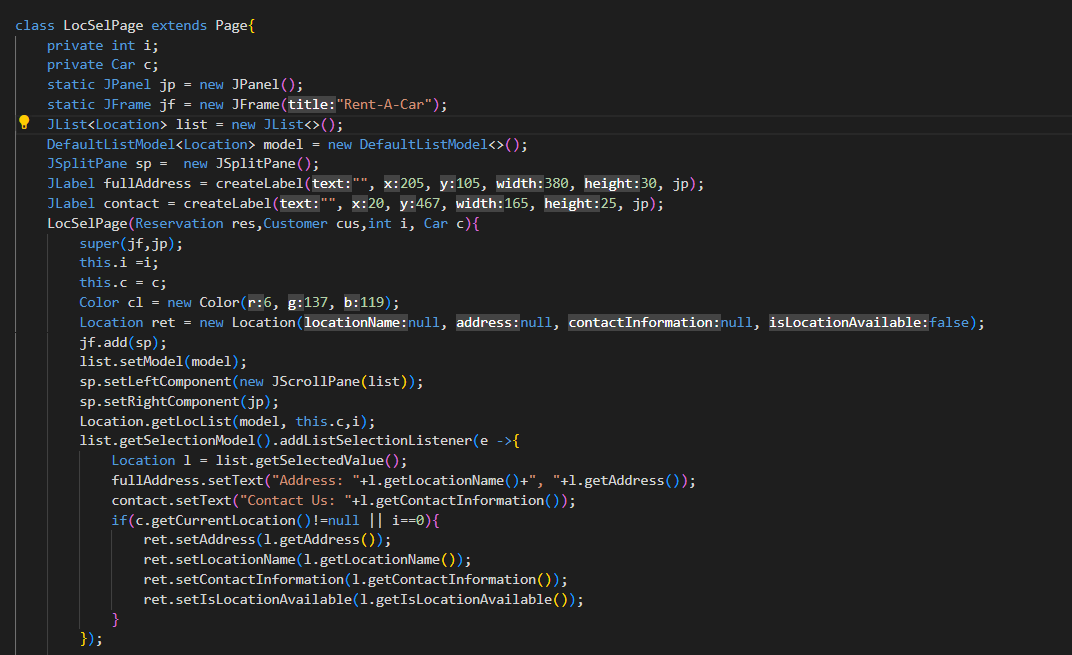
**

*(Figure 14. screenshot of background code of carSelPage class constructor)*

After selecting the location, she/he presses the button to select the drop off location and is directed to the drop off location page.

**

*(Figure 15. screenshot of location selection page)*

**

*(Figure 16. screenshot of background code of LocSelPage class constructor)*

*A screen shot of a computer program

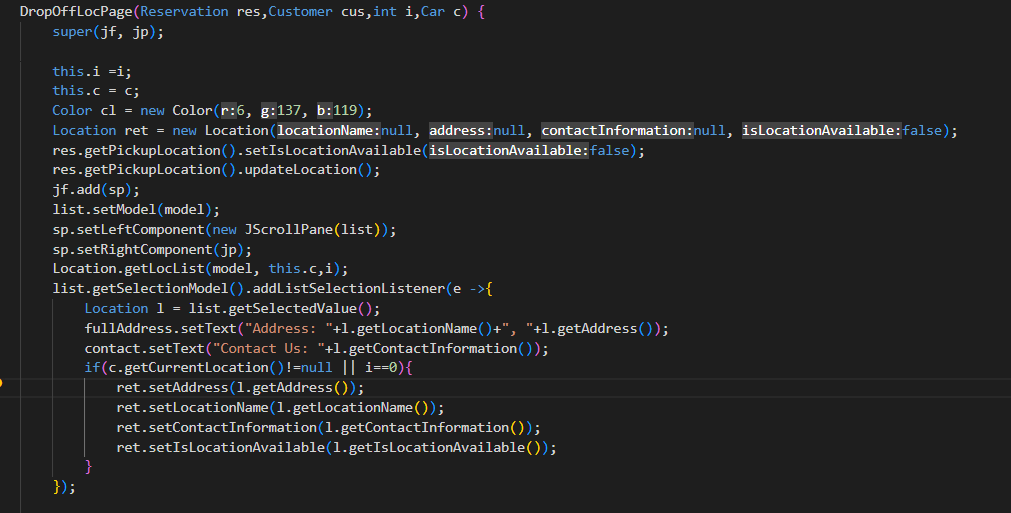
Description automatically generated with low confidence*

*(Figure 15. screenshot of background code of carSelPage class constructor)*

If she/he clicks on the make reservation button on the drop off page, she/he will be directed to the reservation page. Here, all the information about the reservation is given to the user and she/he creates a reservation by clicking the make reservation button.

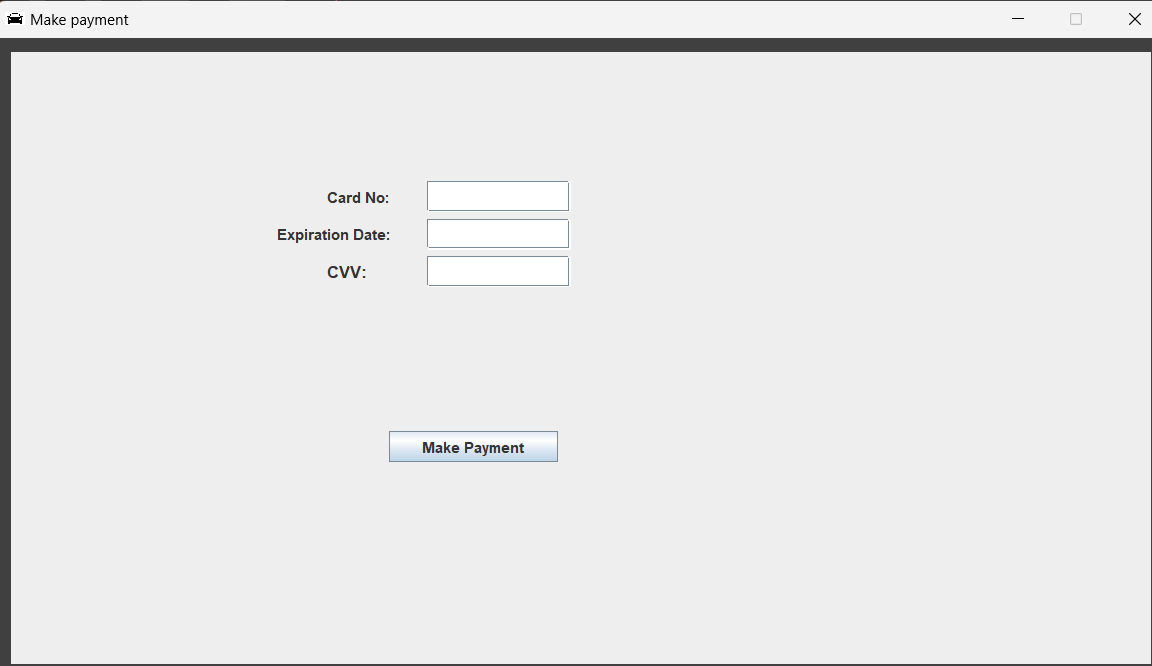
A screenshot of a computer

Description automatically generated *(Figure 16. screenshot of drop off selection page)*

**

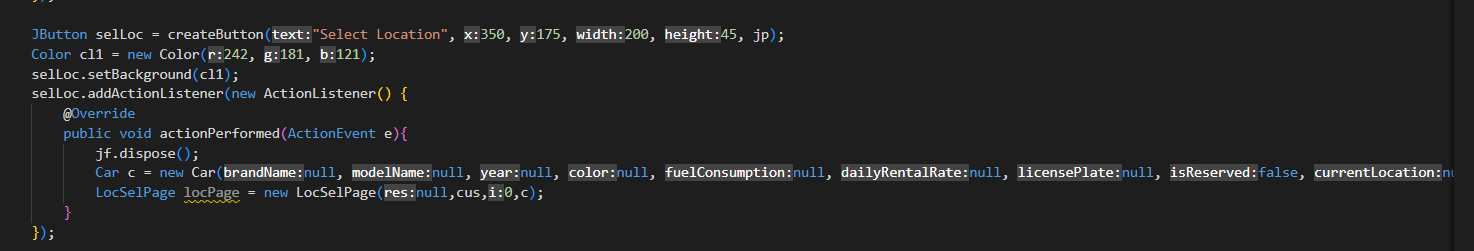
*(Figure 17. screenshot of background code of DropOffLocPage class constructor)*

Directed to the payment page. Here she/he makes a payment by entering her/his card information.

**

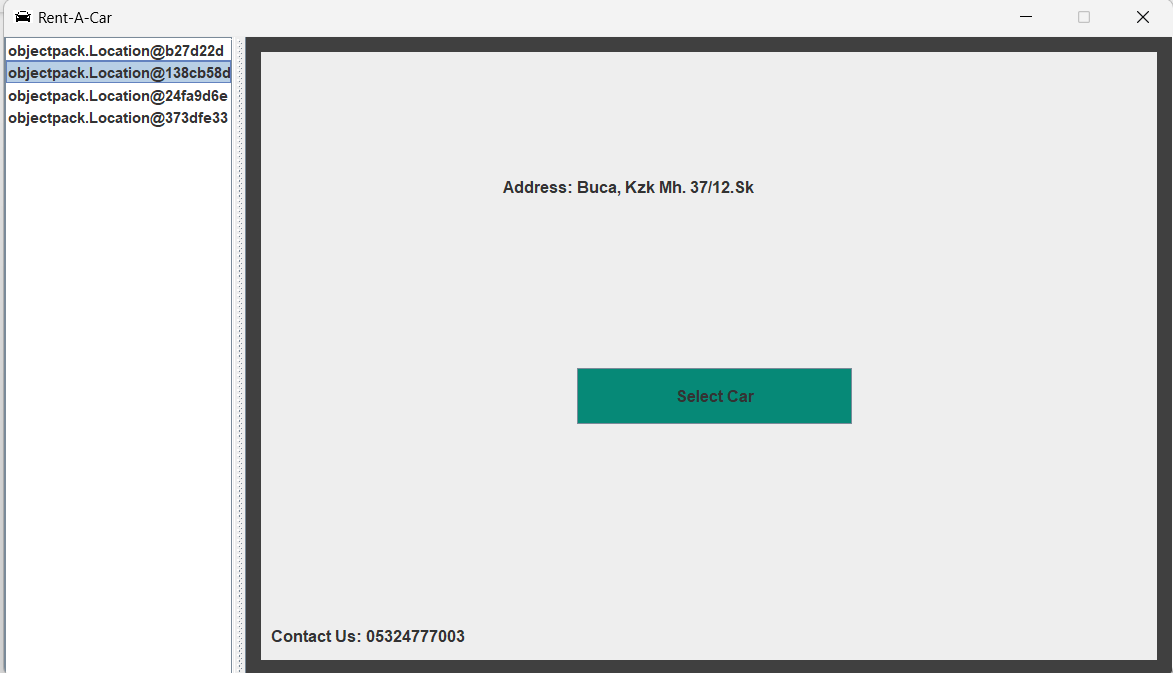
*(Figure 18. screenshot payment page)*

**If the customer clicks on the select location button on the app page,** she/he will be directed to the location selection page.

**

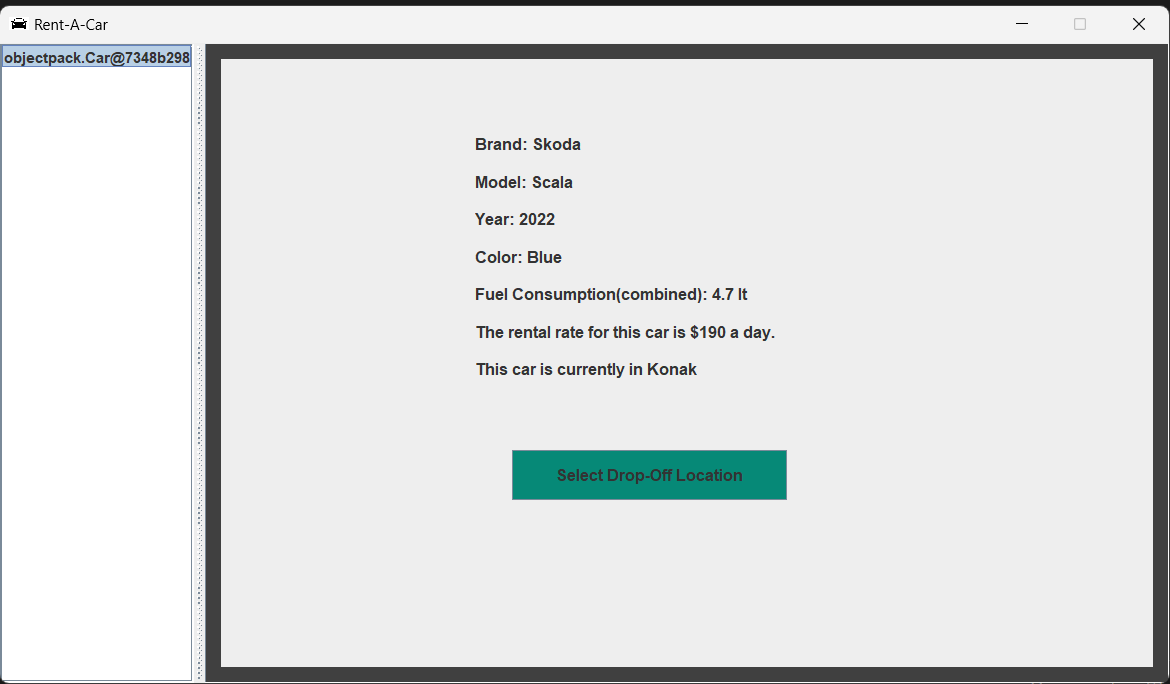
*(Figure 19. screenshot of background code of appPage class)*

She/He sees the information of the location she/he has chosen on the location page and can press the select car button to see the cars of the location she/he has chosen.

**

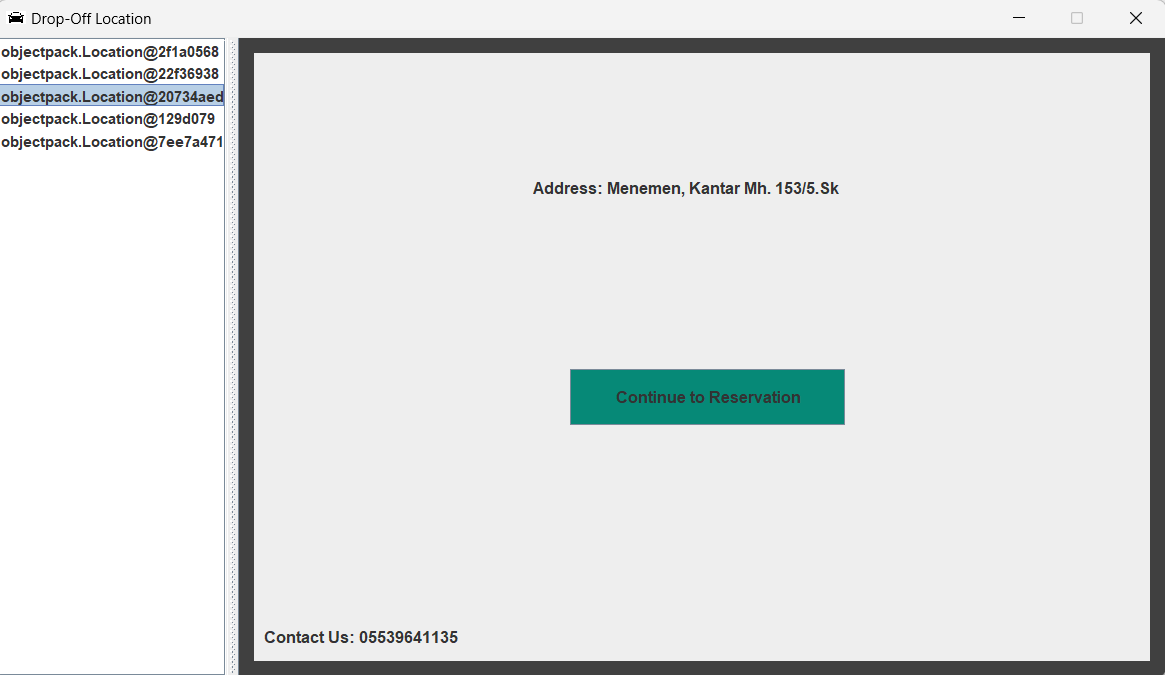
*(Figure 20. screenshot of location selection page)*

After selecting the car, she/he presses the button to select the drop off location and is directed to the drop off location page.

**

*(Figure 21. screenshot of car selection page)*

If she/he clicks on the make reservation button on the drop off page, she/he will be directed to the reservation page. Here, all the information about the reservation is given to the user and she/he creates a reservation by clicking the continue to reservation button.

**

*(Figure 21. screenshot of drop off location selection page)*

*A screen shot of a computer program

Description automatically generated with low confidence*

*(Figure 22. screenshot of background code of drop off loc page class)*

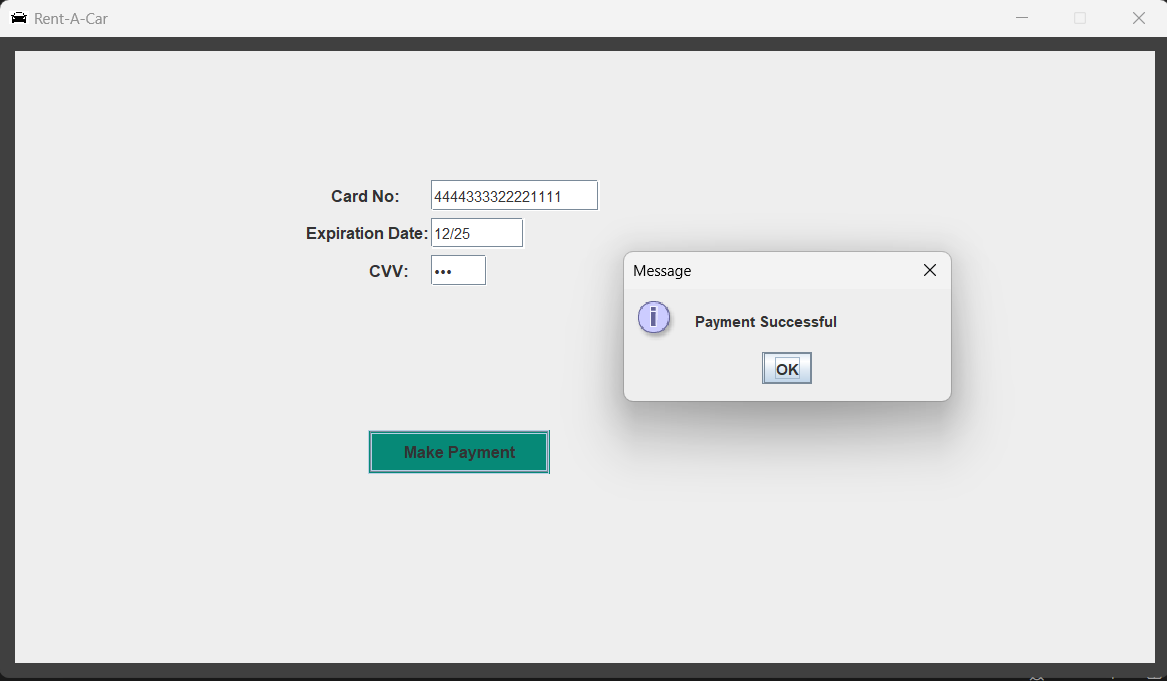
First she/he is shown the reservation history in reservation page, after she/he clicks the make reservation button she/he directed to the payment page

*A screenshot of a computer

Description automatically generated*

*(Figure 22. screenshot reservation page)*

After clicking the make reservation button, you are directed to the payment class. Here, after entering the card information suitable for validation, a successful login notification is made, the reservation is completed and the user is directed to the first login page.

**

*(Figure 23. screenshot payment page)*

*A screen shot of a computer program

Description automatically generated with low confidence*

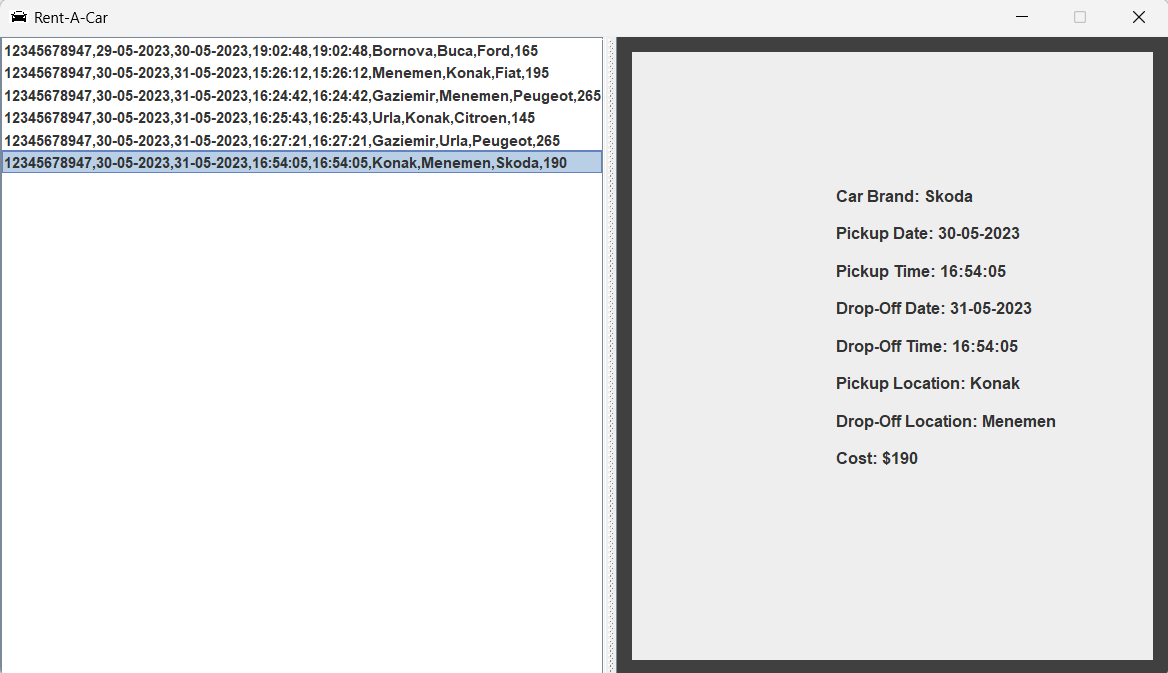
*(Figure 24. screenshot of background code of payment page class)*

**If the customer clicks on your reservation history button**, she/he will be directed to the reservation history page. And here she/he can see all reservations that she/he did in her/his account.

A picture containing text, screenshot, font

Description automatically generated

*(Figure 25. screenshot of background code of app page class)*



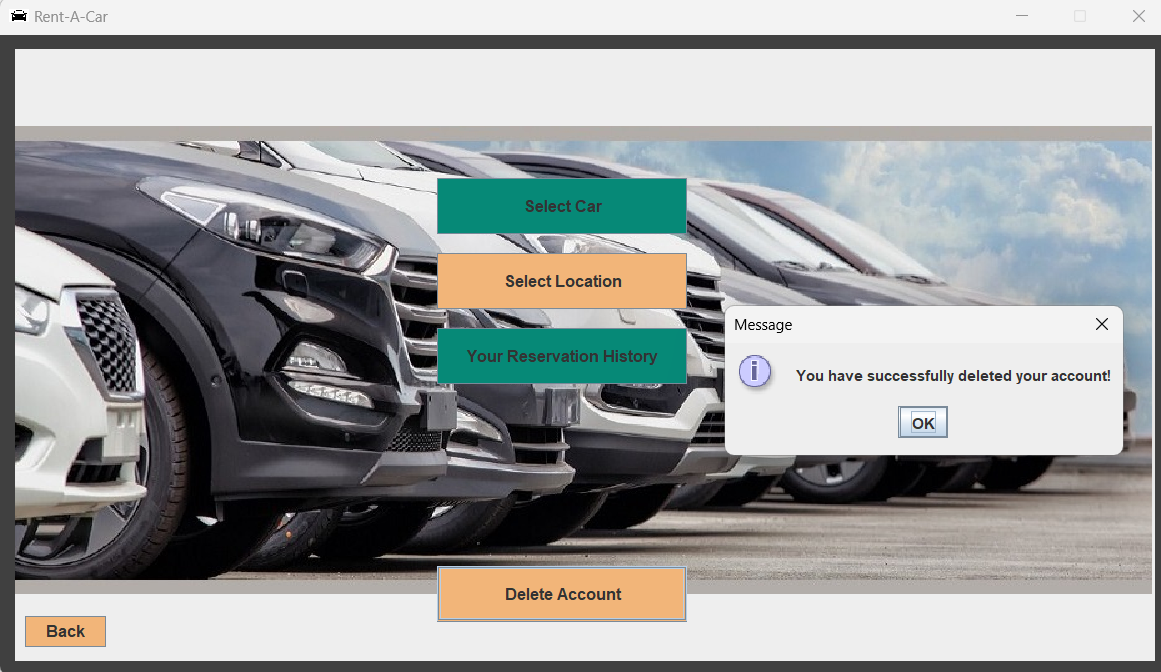
*(Figure 26. screenshot reservation history page)*

A screen shot of a computer program

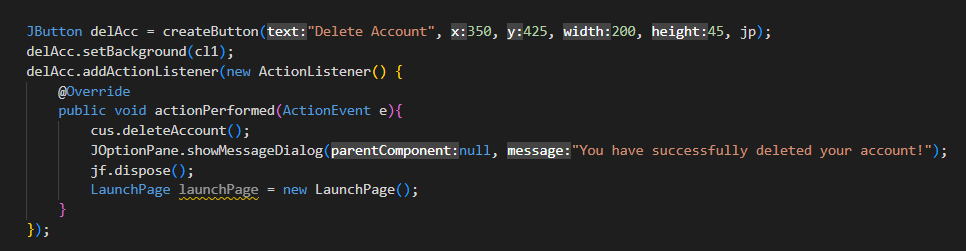
Description automatically generated with medium confidence

*(Figure 27. screenshot of background code of resHis page class)*

**If the customer clicks on delete account button**, she/he will be received a message and her/his account will be deleted.



*(Figure 28. screenshot of app page for delete account operation)*

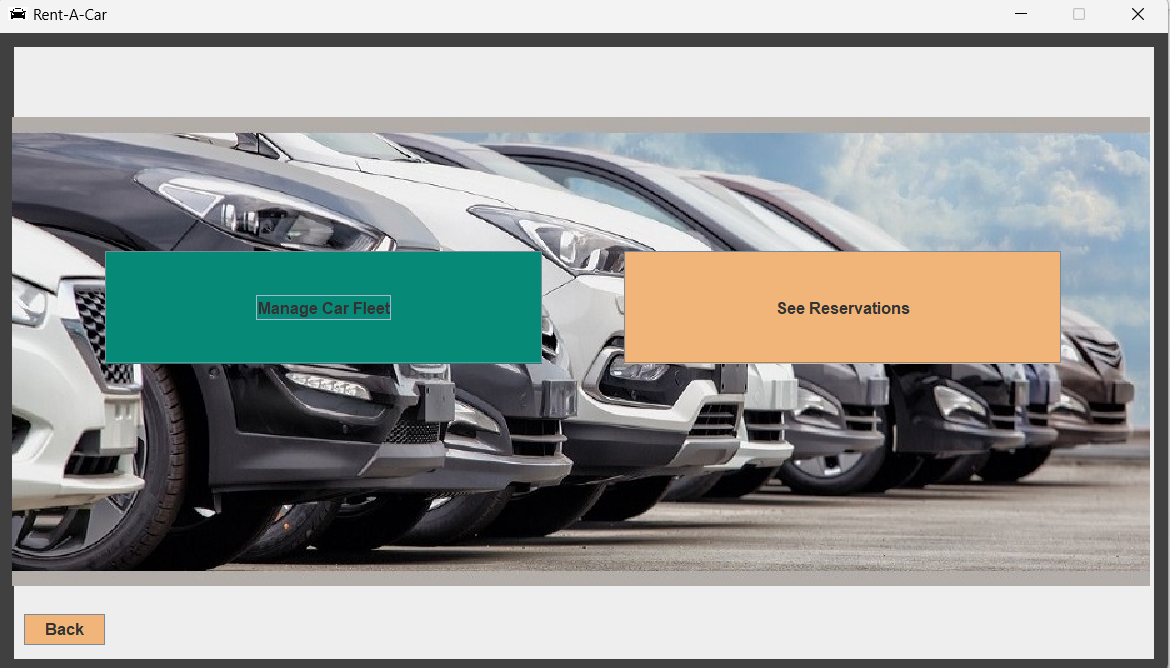


*(Figure 29. screenshot of background code of appPage class)*

**

*(Figure 30. screenshot of background code of customer class)*

**If the admin is logged in**, (admin has a specific email and password), it redirected to the admin page and here two options appear. These are manage car fleet and see reservations.



*(Figure 31. screenshot of admin page)*

A picture containing text, font, software, multimedia software

Description automatically generated

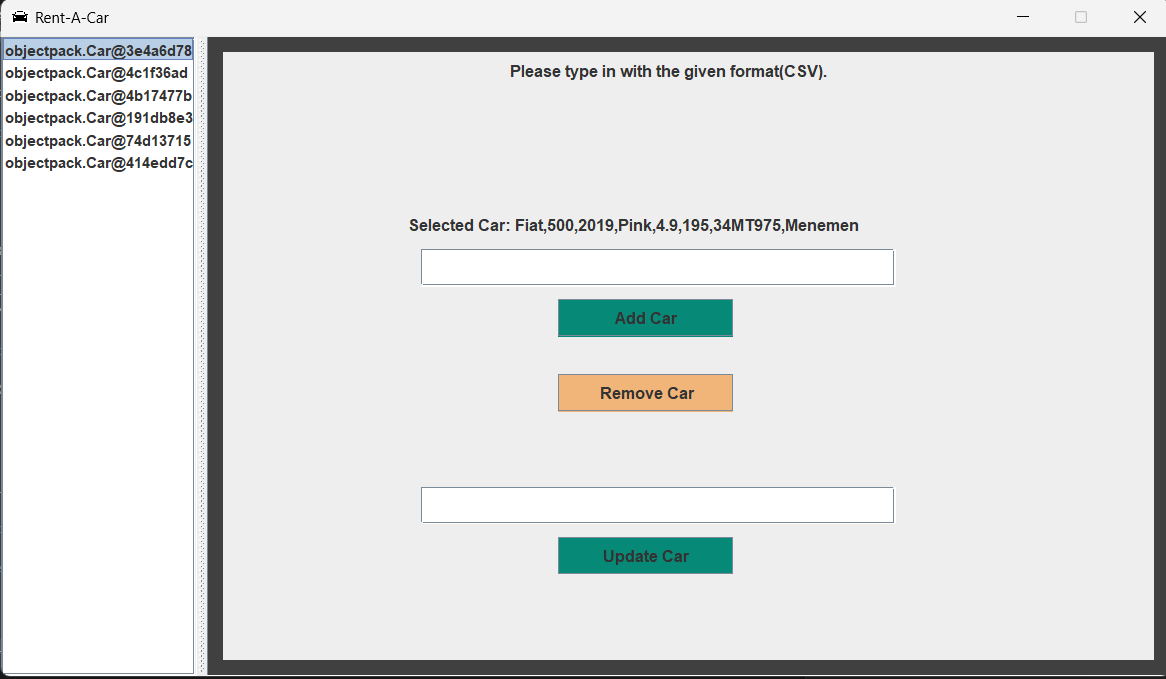
*(Figure 32. screenshot of background code of launch page class)*

**If he/she clicks on the manage car fleet button,** he is directed to the fleet page. Here, the admin can add, delete or update the selected car's location and availability information.

A screen shot of a computer program

Description automatically generated with low confidence

*(Figure 33. screenshot of background code of admin page class)*



*(Figure 34. screenshot of fleet page)*

*A screen shot of a computer program

Description automatically generated with low confidence*

*(Figure 35. screenshot of background code of fleet page class)*

**If he/she clicks on the see reservations button,** he is directed to the reservation history page. Here, the admin can see the all reservations.

*A screen shot of a computer code

Description automatically generated with low confidence*

*(Figure 36. screenshot of background code of admin page class)*

*A screenshot of a computer

Description automatically generated with medium confidence*

*(Figure 37. screenshot of resHistory page)*

# CHAPTER FIVE

CONCLUSION AND FUTURE WORKS

Our project was completed as we wanted. User and admin logins, actions to be taken (selecting a car, choosing a location, making a payment, viewing reservation etc). GUI also completed successfully. We did good time management and completed the homework on time.

This project is very open to development. It currently operates as a daily and single-branch car rental site and pays the same fee whenever the user delivers the car. (The fee varies according to the car.) If the project can be developed, a site with multiple branches can be built that can rent a car for a week or more. At the same time, the amount of money paid by the user may vary depending on the amount of gasoline used and the distance to which the car is delivered.