

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 setters, 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<>(): holds 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.

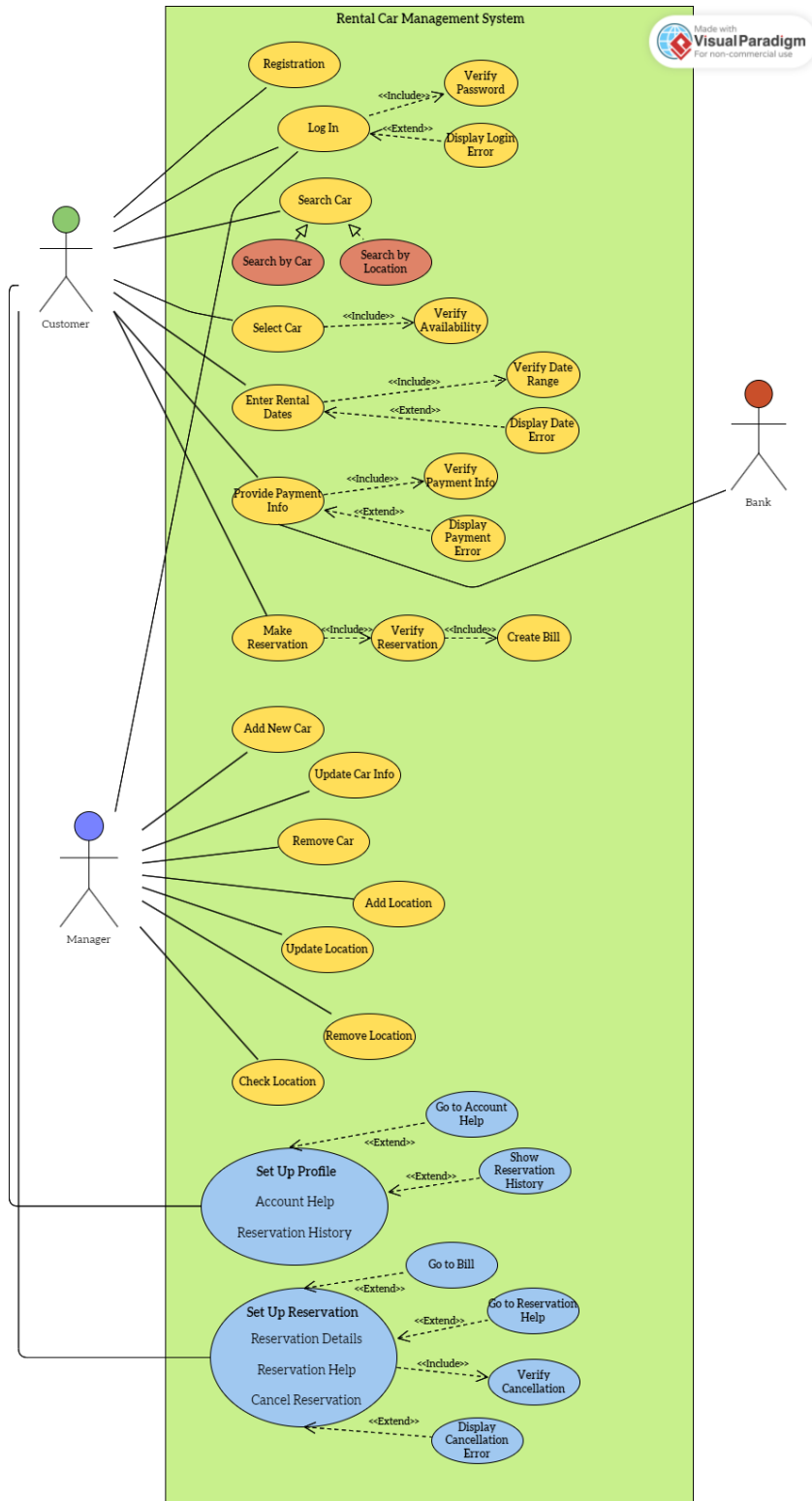
updateLocation(String locationName, String address, Car car): updates the information for an existing location in the system

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

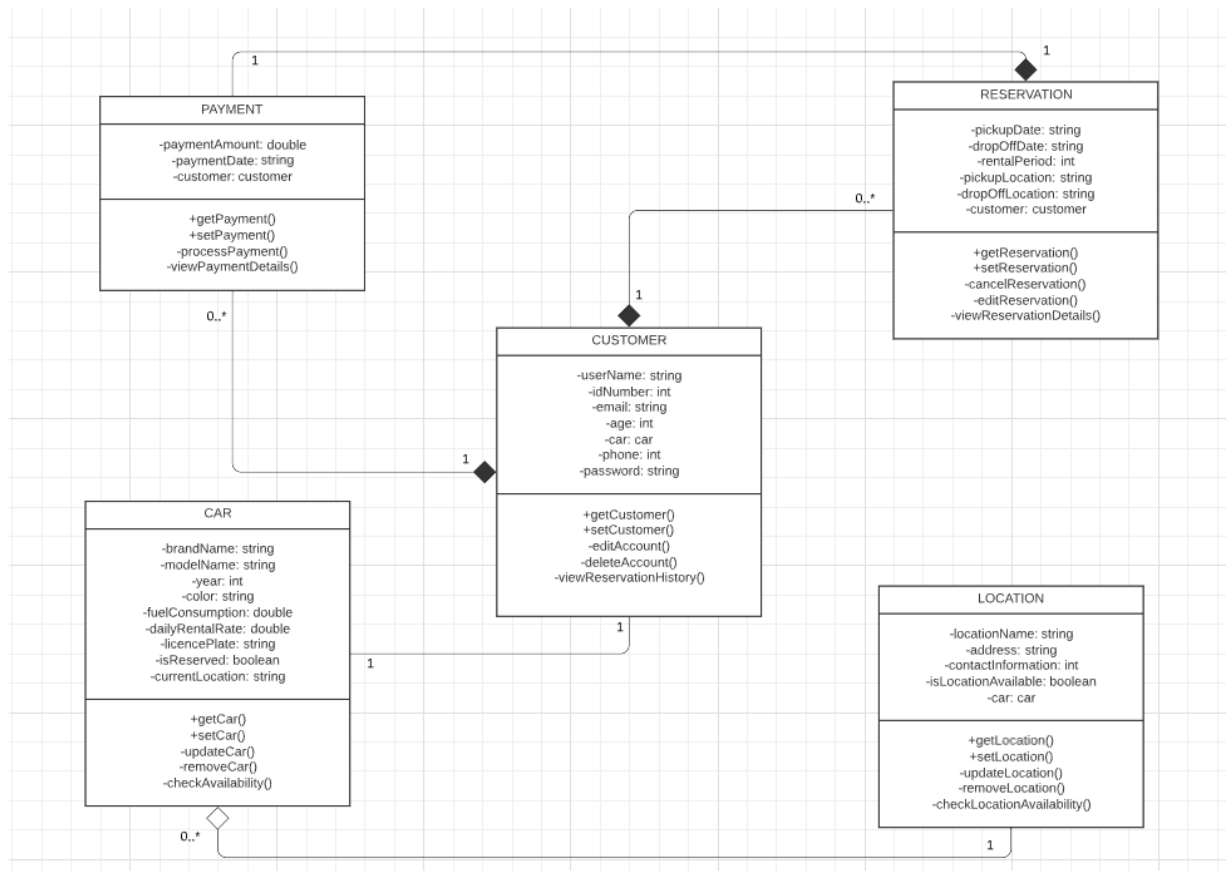
car. The system must check if the selected car is available for rental and whether double booking exists. Therefore, "include" is used. The customer enters the desired rental period. The system must then check if the car is available during the specified rental period. Therefore, "include" is used. If 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.

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.

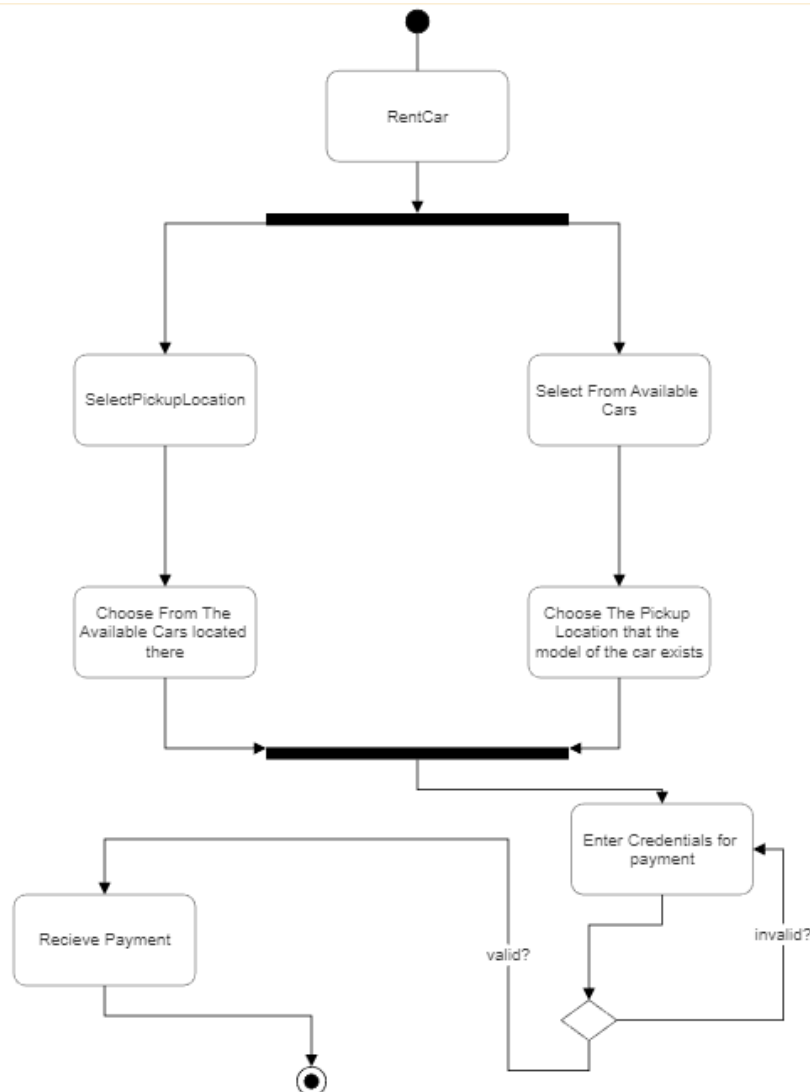
Also, the Payment class is related to the Reservation class as follows:

- 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.

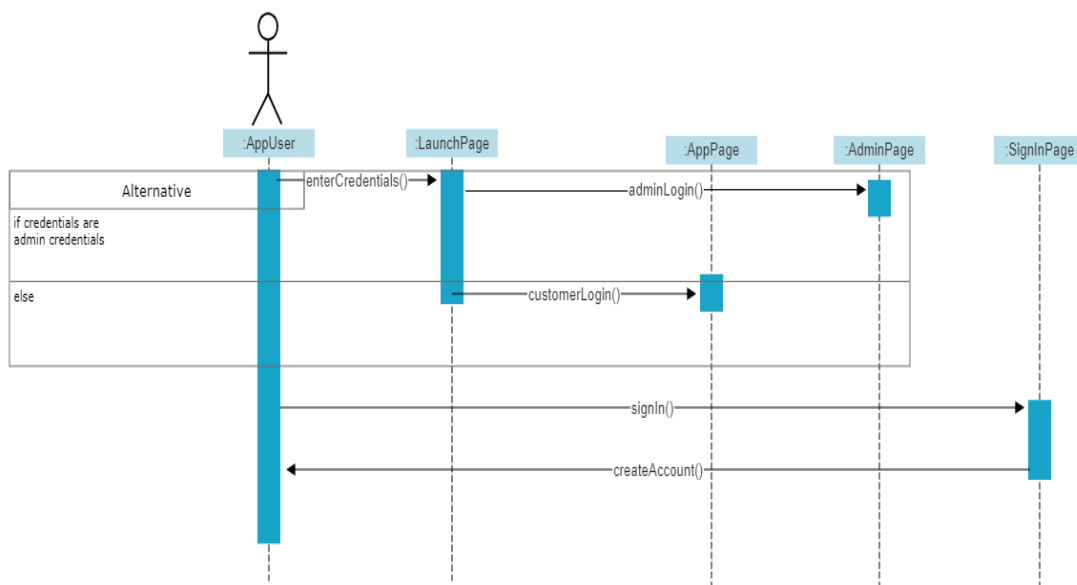
If the customer chooses the car they want first then they are shown a list of the locations that

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

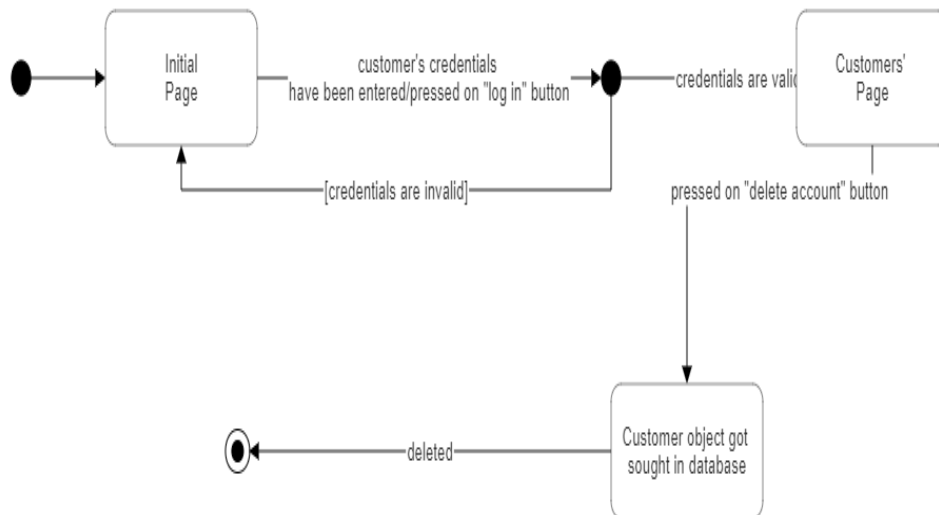


The sequence diagram given above shows the interactions between pages of our app and actor.

As the actor enters their credentials(that exist) on “LoginPage” 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 “LoginPage”.

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.

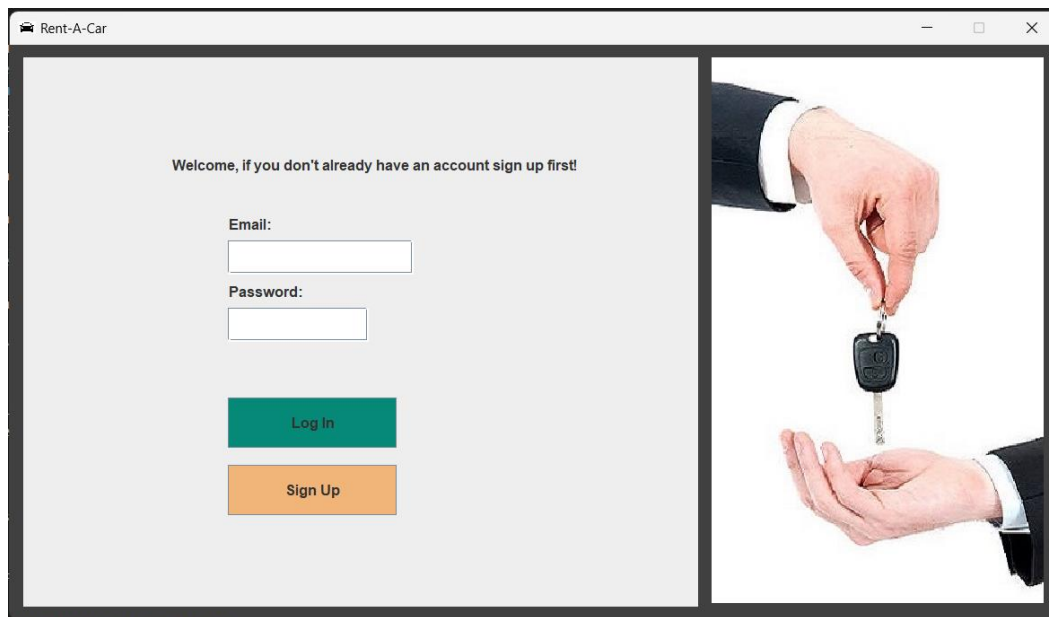
The initial state is our initial page being on the screen, if the customer enters their credentials and presses the “Log In” button we check whether the credentials are valid or invalid. If the credentials are invalid then we return back to the initial state but if they are valid then we change our state 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)

```
public class LaunchPage extends Page{
    static JPanel jp = new JPanel();
    static JFrame jf = new JFrame(title:"Rent-A-Car");
    public LaunchPage(){
        super(jf,jp);

        createLabel(text:"Welcome, if you don't already have an account sign up first!", x:145,y:85 , width:405, height:45,jp);

        createLabel(text:"Email:", x:195, y:145, width:85, height:30,jp);

        JTextField mailField = createField(x:195, y:175, width:165, height:30,jp);

        createLabel(text:"Password:", x:195, y:205, width:85, height:30, jp);

        JPasswordField passwordText = createPasswordField(x:195,y:235,width:125,height:30, jp);

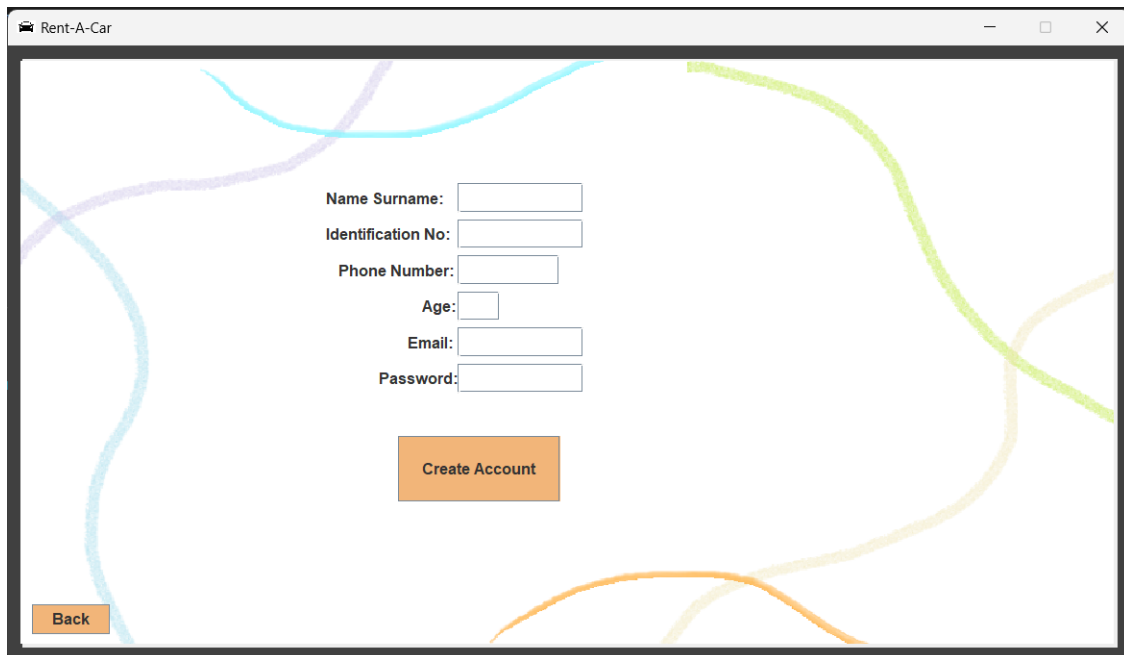
        JButton logIn = createButton(text:"Log In", x:195, y:315, width:150, height:45, jp);
        Color c1 = new Color(r:6, g:137, b:119);
        logIn.setBackground(c1);

        JButton signUp = createButton(text:"Sign Up", x:195, y:375, width:150, height:45, jp);
        Color c = new Color(r:242, g:181, b:121);
        signUp.setBackground(c);

        JLabel failedLogin = createLabel(text:"", x:85, y:465, width:420, height:20, jp);
```

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

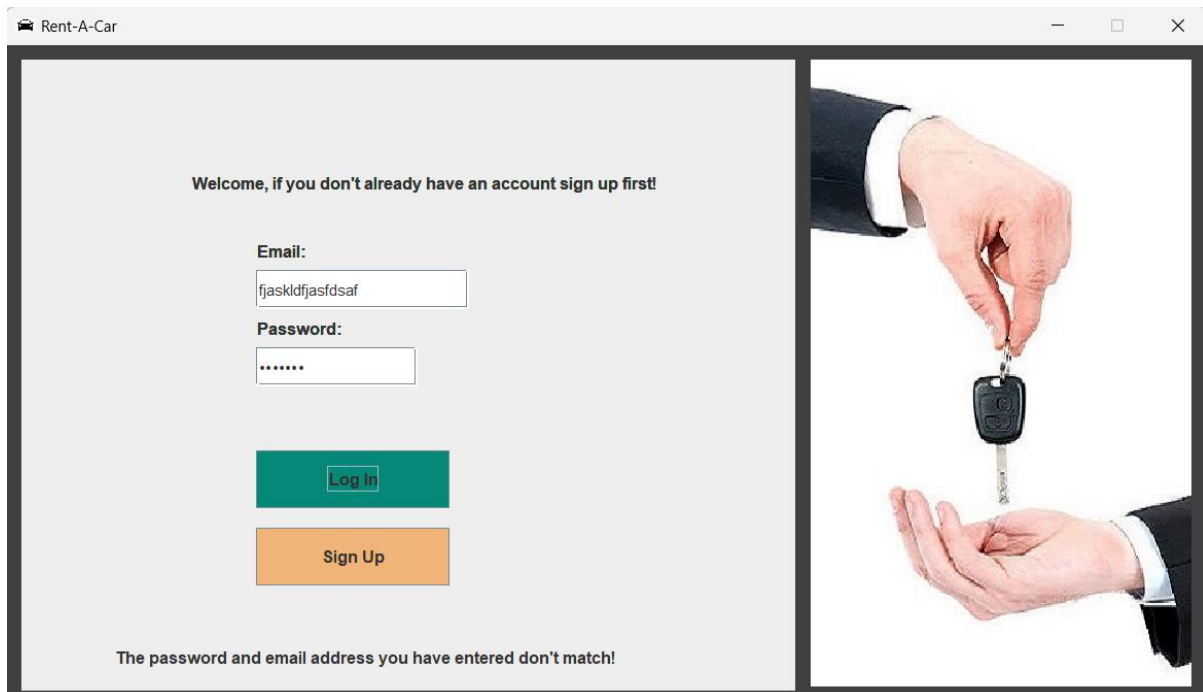
Here, if the user is new, he presses the sign up button. And the registration page appears.



A screenshot of a web browser window titled "Rent-A-Car". The page features a registration form with the following fields: "Name Surname:", "Identification No:", "Phone Number:", "Age:", "Email:", and "Password:". Below the form is an orange button labeled "Create Account". In the bottom left corner, there is a "Back" button. The background of the page is decorated with several overlapping, wavy lines in shades of blue, purple, green, and orange.

(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.



A screenshot of a web browser window titled "Rent-A-Car". The page is divided into two main sections. On the left, there is a login form with the text "Welcome, if you don't already have an account sign up first!". The form includes fields for "Email:" (containing "fjaskldfjasfdsaf") and "Password:" (containing "*****"). Below these fields are two buttons: a green "Log In" button and an orange "Sign Up" button. At the bottom of this section, a message reads: "The password and email address you have entered don't match!". On the right side of the page, there is a photograph of two hands; one hand is holding a set of car keys, and the other hand is reaching out to receive them.

(Figure 4. screenshot of launch page)

```

//When user presses Log In button
logIn.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e){
        String mail = mailField.getText();
        String password = passwordText.getText();
        if(mail.equals(anObject:"admin@rentacar.com")&& password.equals(anObject:"12345")){
            jf.dispose();
            AdminPage adminPage = new AdminPage();
        }
        else if(Customer.doCredentialsMatch(mail, password)){
            jf.dispose();
            Customer cust=Customer.findCustomer(mail);
            AppPage appPage = new AppPage(cust);
        }
        else{
            failedLogin.setText(text:"The password and email address you have entered don't match!");
        }
    }
});

```

(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.

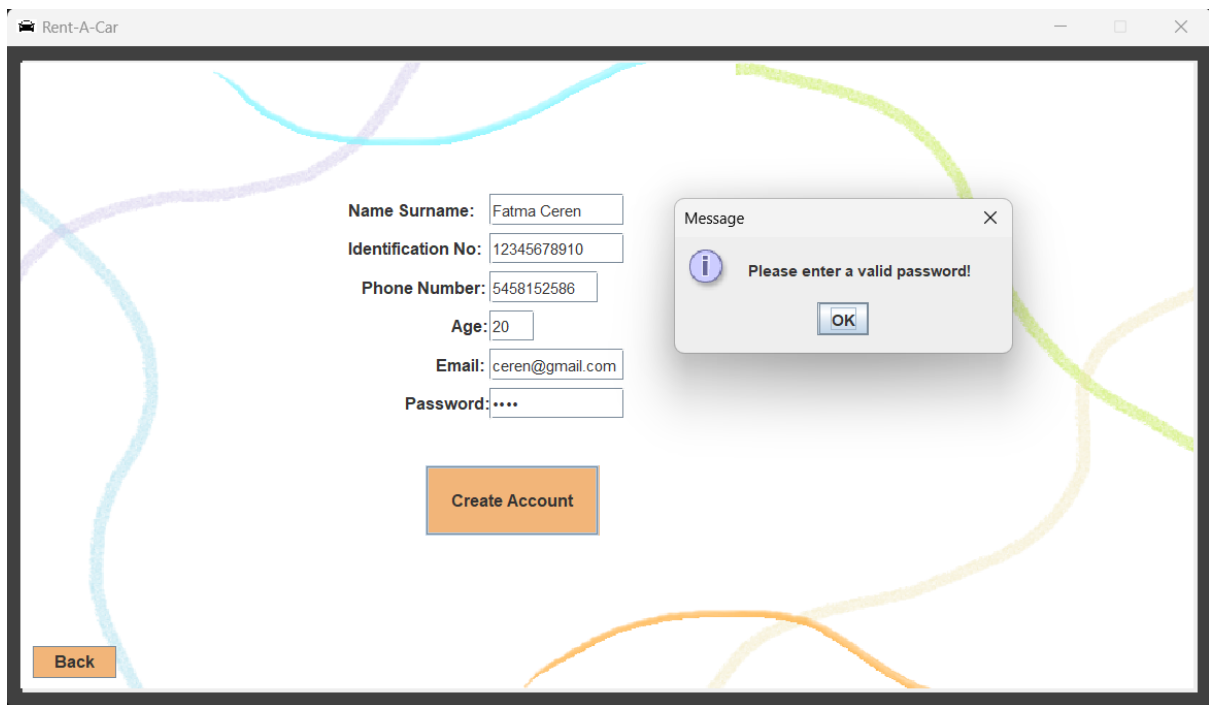
The screenshot shows a web application window titled "Rent-A-Car". Inside, there is a registration form with the following fields and values:

- Name Surname: Fatma Ceren
- Identification No: 12345678910
- Phone Number: 5458152586
- Age: 20
- Email: ceren@gmail.com
- Password:

Below the form is an orange "Create Account" button. In the bottom left corner, there is a "Back" button. A modal message box is open, displaying an information icon and the text "You have successfully created an account!" with an "OK" button.

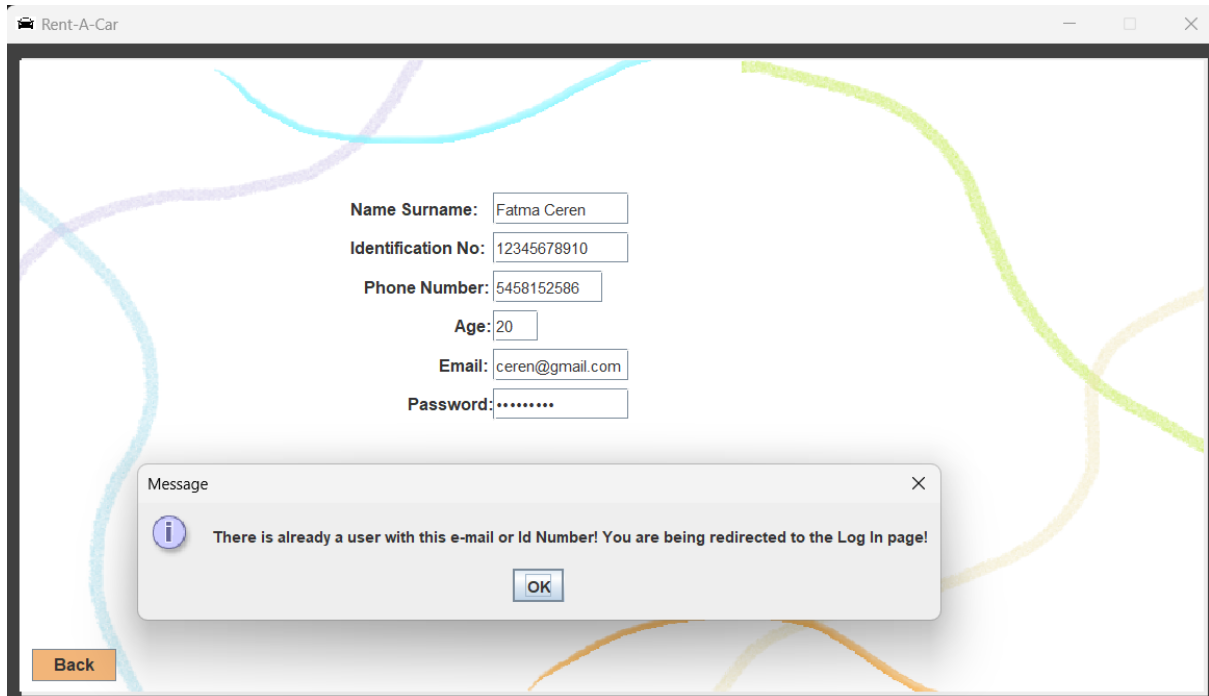
(Figure 6. screenshot of sign up page)

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



(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)

```

SignUpPage(){
    super(jf,jp);

    createLabel(text:"Name Surname:",x:265, y:115,width:105,height:25,jp);

    JTextField nameField = createField(x:375,y:115,width:105,height:25,jp);

    createLabel(text:"Identification No:",x:265, y:145,width:145,height:25,jp);

    JTextField idField = createField(x:375,y:145,width:105,height:25,jp);

    createLabel(text:"Phone Number:",x:275,y:175,width:115,height:25,jp);

    JTextField phoneField = createField(x:375,y:175,width:85,height:25,jp);

    createLabel(text:"Age:",x:345, y:205,width:85,height:25,jp);

    JTextField ageField = createField(x:375,y:205,width:35,height:25,jp);

    createLabel(text:"Email:",x:333,y:235,width:85,height:25,jp);

    JTextField mailField = createField(x:375,y:235,width:105,height:25,jp);

    createLabel(text:"Password:",x:309,y:265,width:85,height:25,jp);

    JPasswordField passwordText = createPasswordField(x:375,y:265,width:105,height:25, jp);

    JButton createAcc = createButton(text:"Create Account",x:325, y:325, width:135, height:55,jp);
    Color cl = new Color(r:242, g:181, b:121);
    createAcc.setBackground(cl);
}

```

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

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

```

createAcc.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e){
        if(nameField.getText()!=null && mailField.getText()!=null && ageField.getText()!=null && idField.getText()!=null && passwordText.getText()!=null && phoneField.getText()!=null){
            Customer newCust = new Customer(nameField.getText(), mailField.getText(), passwordText.getText(), idField.getText(), Integer.parseInt(ageField.getText()), phoneField.getText());
            if(newCust.passwordValidation(newCust) && newCust.toDocument(newCust)){
                JOptionPane.showMessageDialog(parentComponent:null, message:"You have successfully created an account!");
                jf.dispose();
                LaunchPage launchP = new LaunchPage();
            }
            else if(!newCust.userNameValidation(newCust))
                JOptionPane.showMessageDialog(parentComponent:null, message:"Please enter a valid user name!(like John Marker)");
            else if(!newCust.emailValidation(newCust))
                JOptionPane.showMessageDialog(parentComponent:null, message:"Please enter a valid email!");
            else if(!newCust.passwordValidation(newCust))
                JOptionPane.showMessageDialog(parentComponent:null, message:"Please enter a valid password!");
            else if(!newCust.idValidation(newCust))
                JOptionPane.showMessageDialog(parentComponent:null, message:"Please enter a id!");
            else if(!newCust.ageValidation(newCust))
                JOptionPane.showMessageDialog(parentComponent:null, message:"Please enter a valid age! (This app is not for people under 18)");
            else if(!newCust.phoneValidation(newCust))
                JOptionPane.showMessageDialog(parentComponent:null, message:"Please enter a valid phone number!");
            else if(!newCust.toDocument(newCust)){
                JOptionPane.showMessageDialog(parentComponent:null, message:"There is already a user with this e-mail or Id Number! You are being redirected to the Log In page!");
                jf.dispose();
                LaunchPage launchP = new LaunchPage();
            }
        }
        else{
            System.out.println(x:"Error");
        }
    }
}

```

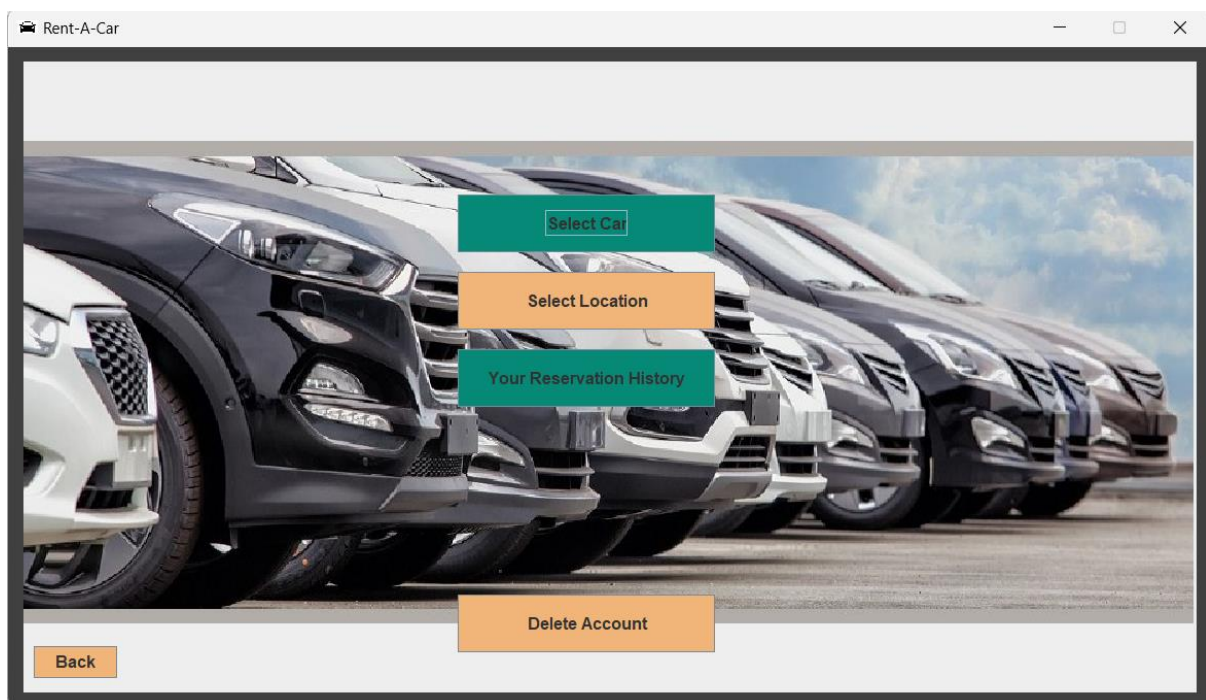
(Figure 10. screenshot of background code of SignUpPage class)

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.

```
//When user presses Log In button
logIn.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e){
        String mail = mailField.getText();
        String password = passwordText.getText();
        if(mail.equals(anObject:"admin@rentacar.com")&& password.equals(anObject:"12345")){
            jf.dispose();
            AdminPage adminPage = new AdminPage();
        }
        else if(Customer.doCredentialsMatch(mail, password)){
            jf.dispose();
            Customer cust=Customer.findCustomer(mail);
            AppPage appPage = new AppPage(cust);
        }
        else{
            failedLogin.setText(text:"The password and email address you have entered don't match!");
        }
    }
});
```

(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)

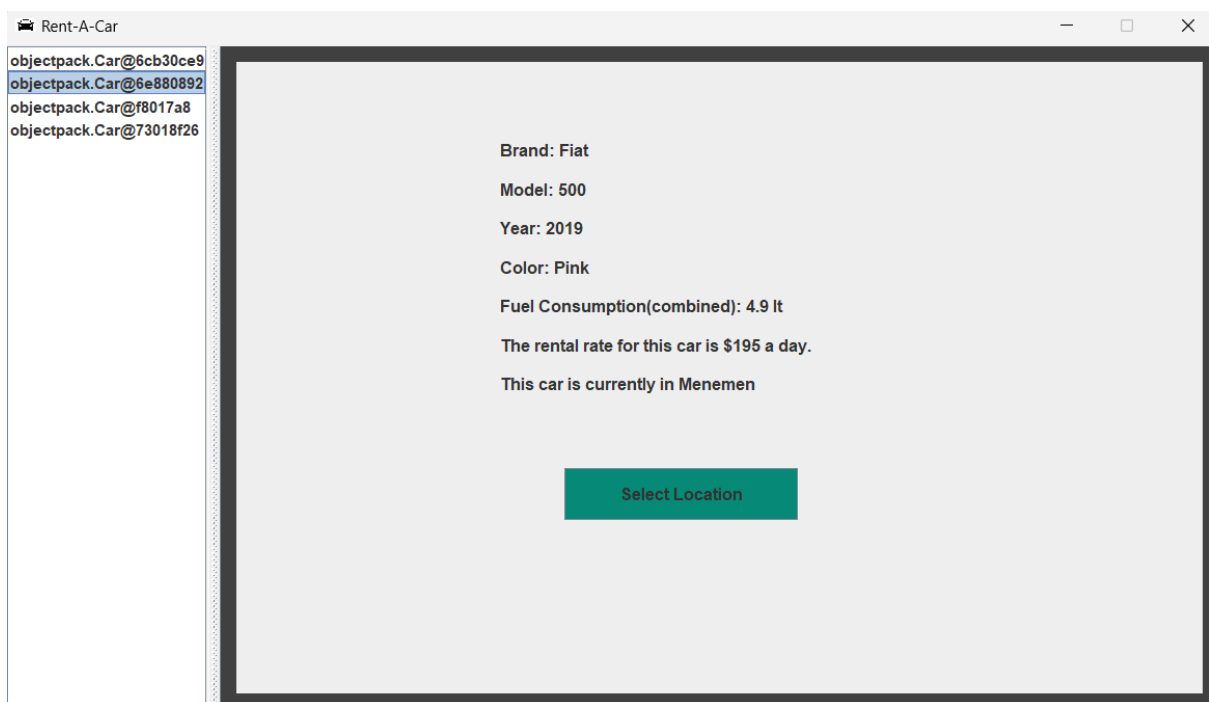
If the customer clicks on the select car button, she/he will be directed to the car selection page.

```
class AppPage extends Page{
    static JPanel jp = new JPanel();
    static JFrame jf = new JFrame(title:"Rent-A-Car");
    AppPage(Customer cus){
        super(jf,jp);

        JButton selCar = createButton(text:"Select Car", x:350, y:115, width:200, height:45, jp);
        Color c1 = new Color(r:6, g:137, b:119);
        selCar.setBackground(c1);
        selCar.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e){
                jf.dispose();
                Location l = new Location(locationName:null, address:null, contactInformation:null, isLocationAvailable:false);
                CarSelPage carPage = new CarSelPage(res:null,cus,i:0,l);
            }
        });
    }
}
```

(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)

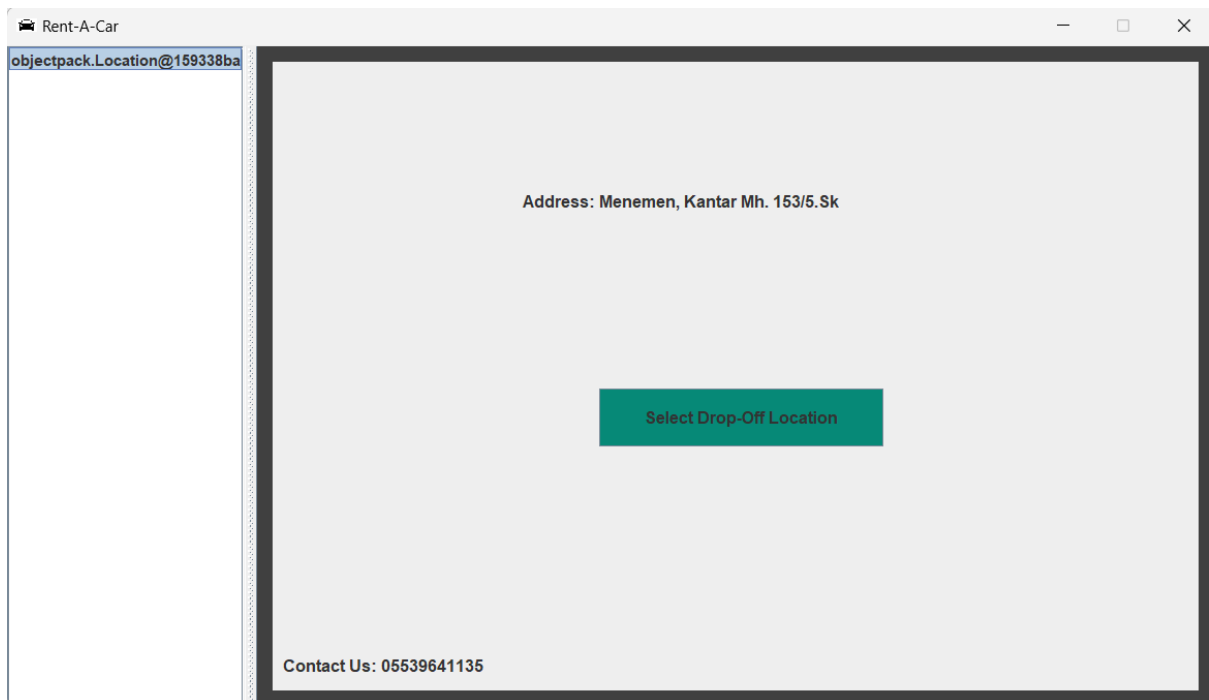
```

CarSelPage(Reservation res, Customer cus, int i, Location l){
    super(jf, jp);
    Color c1 = new Color(r#6, g#137, b#119);
    this.i=i;
    this.l=l;
    Car ret = new Car(brandName:null, modelName:null, year:null, color:null, fuelConsumption:null, dailyRentalRate:null, licensePlate:null, isReserved:false, currentLocation:null);
    jf.add(sp);
    list.setModel(model);
    sp.setLeftComponent(new JScrollPane(list));
    sp.setRightComponent(jp);
    Car.carList(model, this.l);
    list.getSelectionModel().addListSelectionListener(e ->{
        Car c = list.getSelectedValue();
        brand.setText("Brand: "+c.getBrandName());
        carModel.setText("Model: "+c.getModelName());
        year.setText("Year: "+c.getYear());
        color.setText("Color: "+c.getColor());
        fuelCons.setText("Fuel Consumption(Combined): "+c.getFuelConsumption()+" lt");
        rental.setText("The rental rate for this car is $" + c.getDailyRentalRate()+" a day.");
        location.setText("This car is currently in "+c.getCurrentLocation());
        if(this.l.getLocationName()!=null || i==0){
            ret.setBrandName(c.getBrandName());
            ret.setModelName(c.getModelName());
            ret.setYear(c.getYear());
            ret.setColor(c.getColor());
            ret.setFuelConsumption(c.getFuelConsumption());
            ret.setDailyRentalRate(c.getDailyRentalRate());
            ret.setLicensePlate(c.getLicensePlate());
            ret.setCurrentLocation(c.getCurrentLocation());
            ret.setIsReserved(c.getIsReserved());
        }
    });
}

```

(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)

```

class LocSelPage extends Page{
    private int i;
    private Car c;
    static JPanel jp = new JPanel();
    static JFrame jf = new JFrame(title:"Rent-A-Car");
    JList<Location> list = new JList<>();
    DefaultListModel<Location> model = new DefaultListModel<>();
    JSplitPane sp = new JSplitPane();
    JLabel fullAddress = createLabel(text:"", x:205, y:105, width:380, height:30, jp);
    JLabel contact = createLabel(text:"", x:20, y:467, width:165, height:25, jp);
    LocSelPage(Reservation res, Customer cus, int i, Car c){
        super(jf, jp);
        this.i = i;
        this.c = c;
        Color cl = new Color(r:6, g:137, b:119);
        Location ret = new Location(locationName:null, address:null, contactInformation:null, isLocationAvailable:false);
        jf.add(sp);
        list.setModel(model);
        sp.setLeftComponent(new JScrollPane(list));
        sp.setRightComponent(jp);
        Location.getLocList(model, this.c, i);
        list.getSelectionModel().addListSelectionListener(e ->{
            Location l = list.getSelectedValue();
            fullAddress.setText("Address: "+l.getLocationName()+", "+l.getAddress());
            contact.setText("Contact Us: "+l.getContactInformation());
            if(c.getCurrentLocation()!=null || i==0){
                ret.setAddress(l.getAddress());
                ret.setLocationName(l.getLocationName());
                ret.setContactInformation(l.getContactInformation());
                ret.setIsLocationAvailable(l.getIsLocationAvailable());
            }
        });
    }
}

```

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

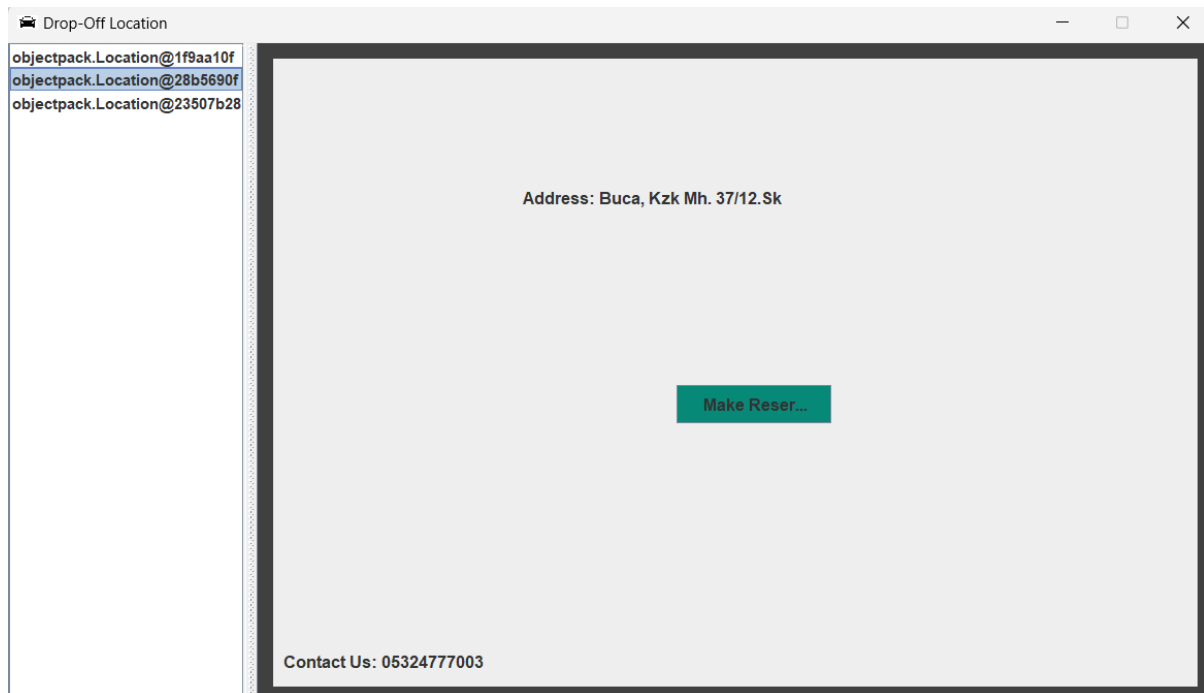
```

});
if(i==0){
    JButton selLoc = createButton(text:"Select Location", x:265, y:325, width:180, height:40, jp);
    selLoc.setBackground(cl);
    selLoc.addActionListener((ActionListener) new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e){
            jf.dispose();
            Reservation res = new Reservation(pickupLocation:null, dropOffLocation:null, ret);
            LocSelPage locPage = new LocSelPage(res, cus, i+1, ret);
        }
    });
}
else{
    res.setCar(ret);
    l.setIsLocationAvailable(isLocationAvailable:false);
    l.updateLocation();
    JButton slcLoc = createButton(text:"Select Drop-Off Location", x:245, y:325, width:220, height:40, jp);
    slcLoc.setBackground(cl);
    slcLoc.addActionListener((ActionListener) new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e){
            jf.dispose();
            DropOffLocPage dLocPage = new DropOffLocPage(res, cus, i+2, ret);
        }
    });
}
}

```

(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.



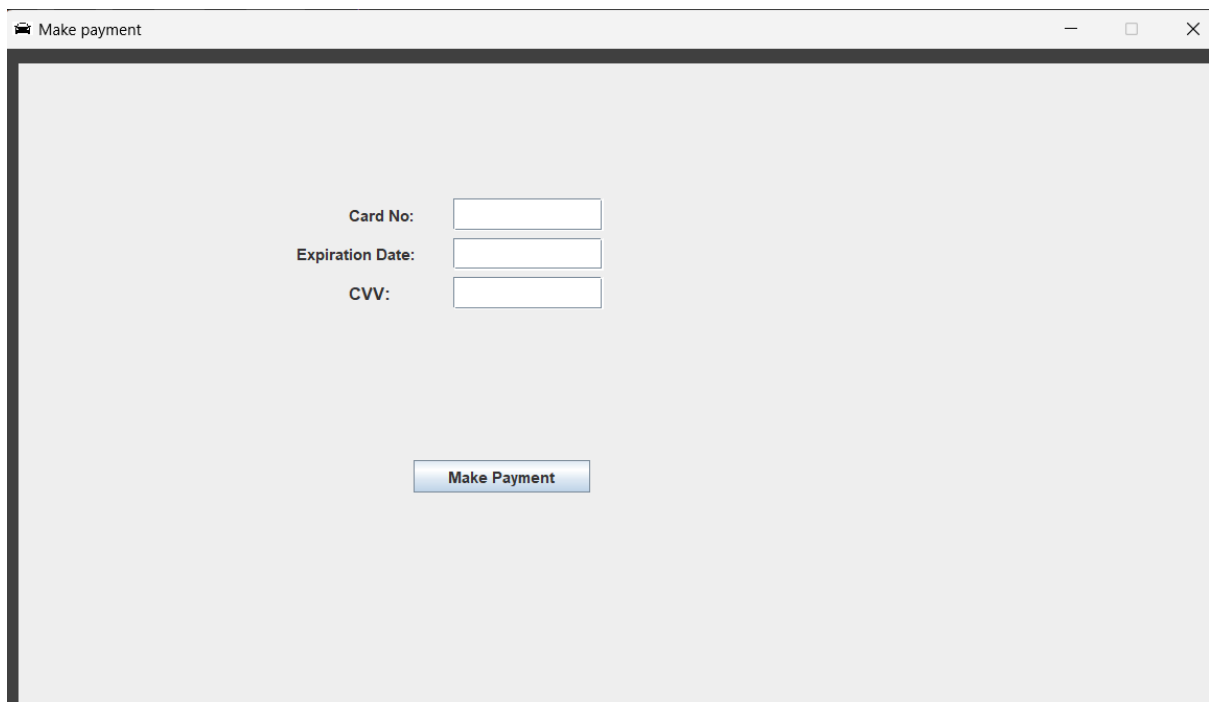
(Figure 16. screenshot of drop off selection page)

```
DropOffLocPage(Reservation res, Customer cus, int i, Car c) {
    super(jf, jp);

    this.i = i;
    this.c = c;
    Color cl = new Color(R:6, G:137, B:119);
    Location ret = new Location(locationName:null, address:null, contactInformation:null, isLocationAvailable:false);
    res.getPickupLocation().setIsLocationAvailable(isLocationAvailable:false);
    res.getPickupLocation().updateLocation();
    jf.add(sp);
    list.setModel(model);
    sp.setLeftComponent(new JScrollPane(list));
    sp.setRightComponent(jp);
    location.getLocList(model, this.c, i);
    list.getSelectionModel().addListSelectionListener(e ->{
        Location l = list.getSelectedValue();
        fullAddress.setText("Address: "+l.getLocationName()+" ", "+l.getAddress());
        contact.setText("Contact Us: "+l.getContactInformation());
        if(c.getCurrentLocation()!=null || i==0){
            ret.setAddress(l.getAddress());
            ret.setLocationName(l.getLocationName());
            ret.setContactInformation(l.getContactInformation());
            ret.setIsLocationAvailable(l.getIsLocationAvailable());
        }
    });
}
```

(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.

A screenshot of a Java Swing window titled "Make payment". The window has a light gray background and a dark gray border. Inside, there are three text input fields arranged vertically. The first field is labeled "Card No:", the second "Expiration Date:", and the third "CVV:". Below these fields is a blue button with the text "Make Payment".

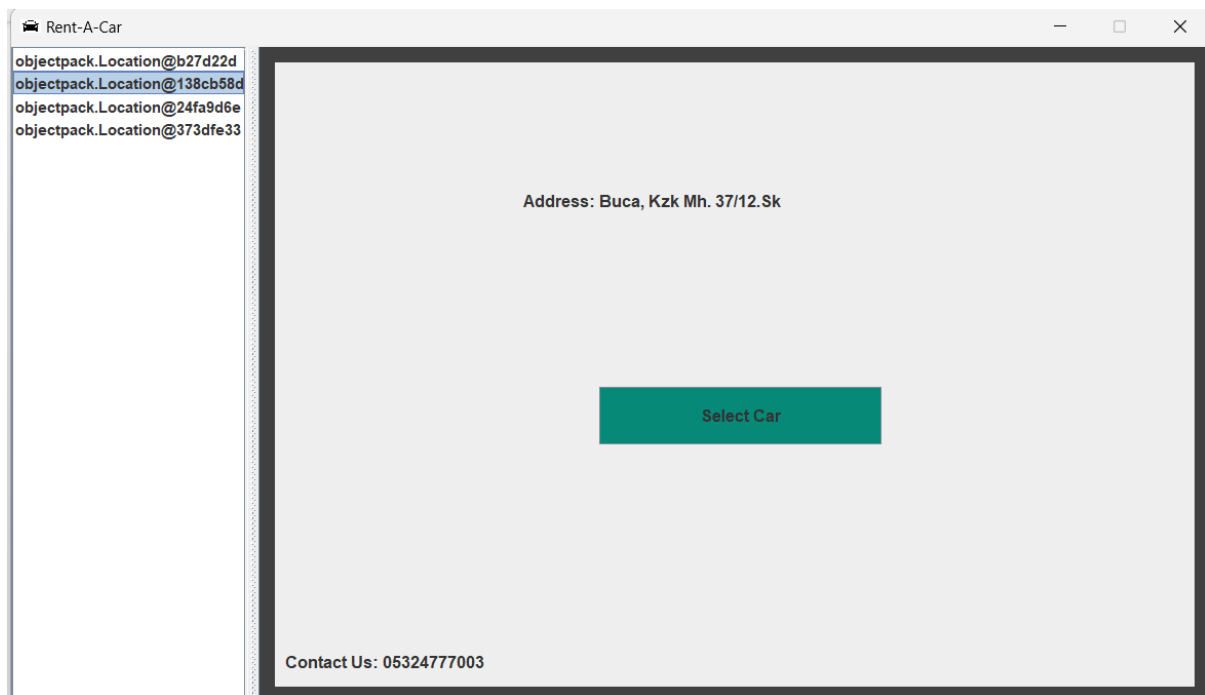
(Figure 18. screenshot payment page)

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

```
777 JButton selloc = createButton(text:"Select Location", x:350, y:175, width:200, height:45, jp);
778 Color c11 = new Color(r:242, g:181, b:121);
779 selloc.setBackground(c11);
780 selloc.addActionListener(new ActionListener() {
781     @Override
782     public void actionPerformed(ActionEvent e){
783         jf.dispose();
784         Car c = new Car(brandName:null, modelName:null, year:null, color:null, fuelConsumption:null, dailyRentalRate:null, licensePlate:null, isReserved:false, currentLocation:n
785         LocSelPage locPage = new LocSelPage(res:null,cus,i:0,c);
786     }
787 });
```

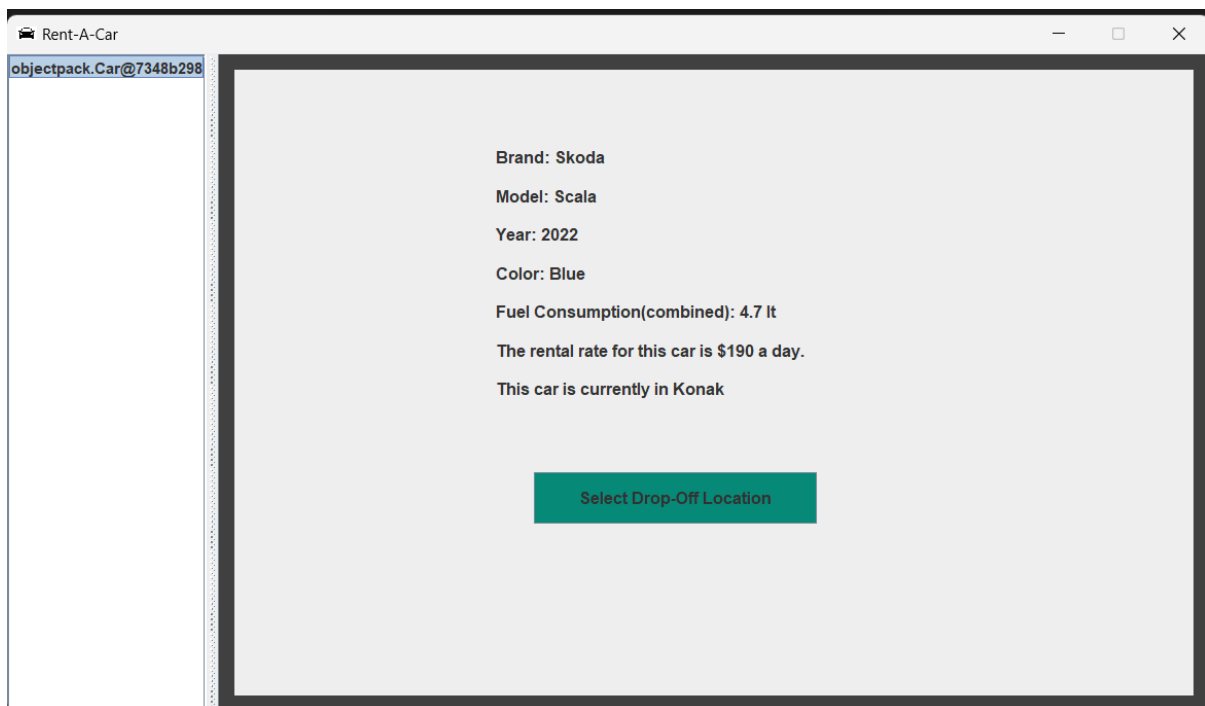
(Figure 19. screenshot of background code of appPage class)

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.



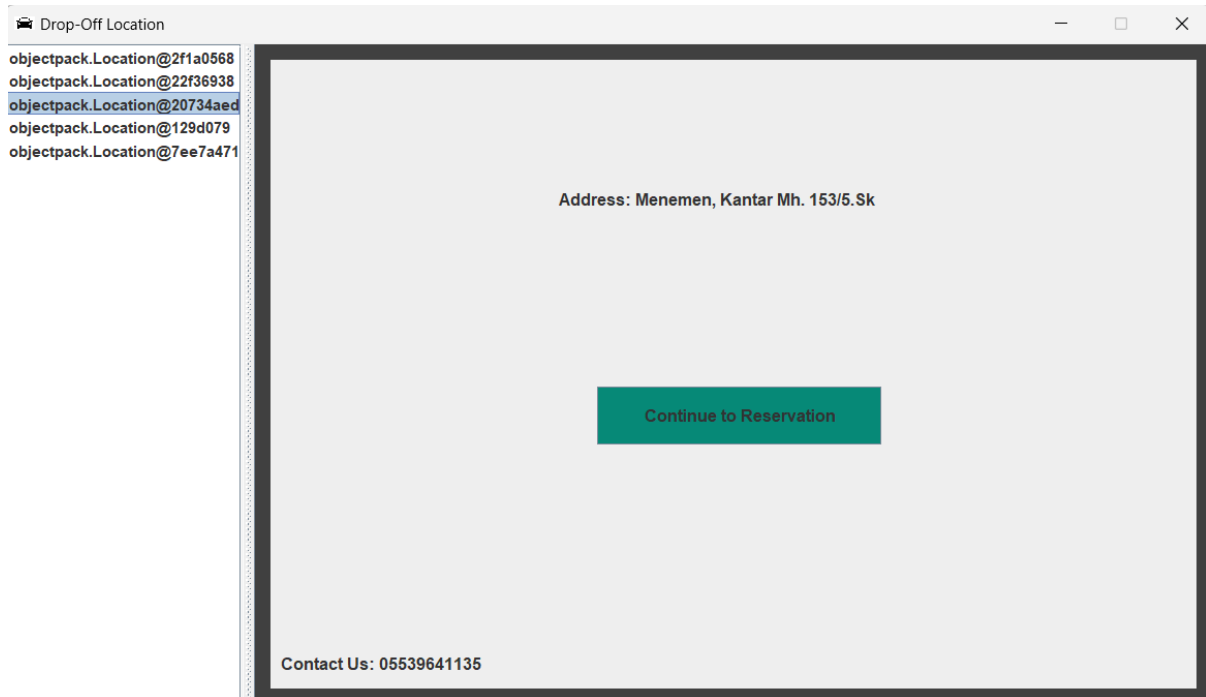
(Figure 20. screenshot of location selection page)

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.

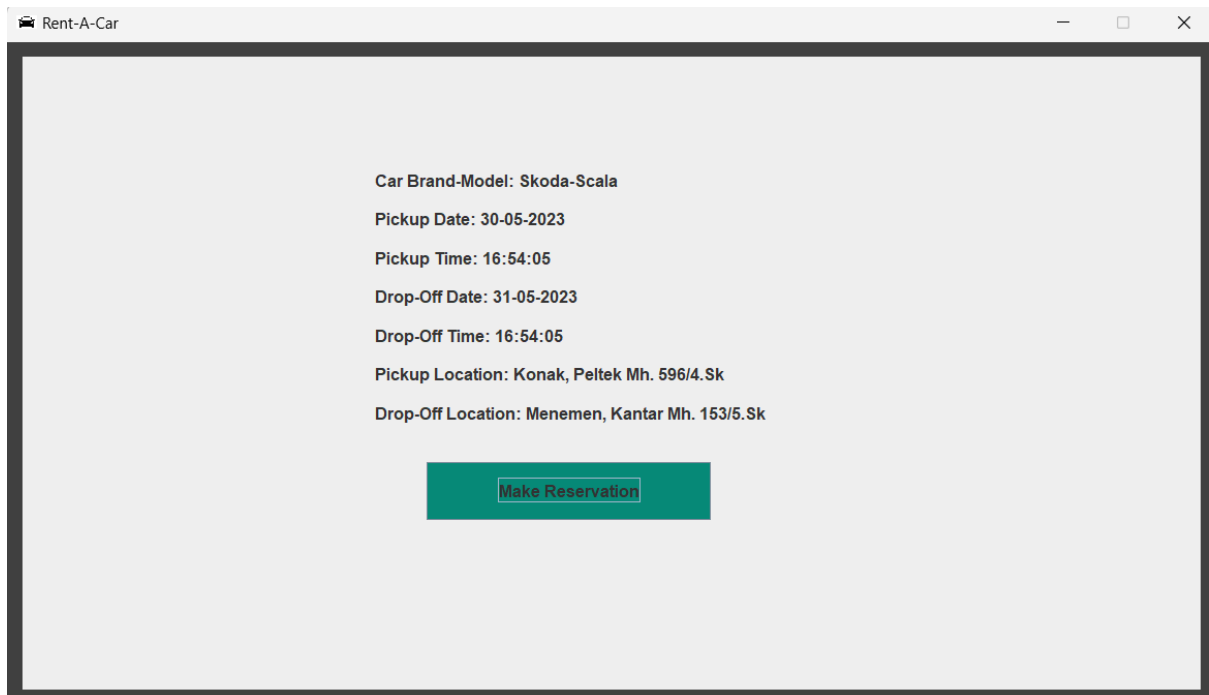


(Figure 21. screenshot of drop off location selection page)

```
res.setDropOffLocation(ret);
JButton makeRes = createButton(text:"Continue to Reservation", x:265, y:265, width:220, height:45, jp);
makeRes.setBackground(c1);
makeRes.addActionListener((ActionListener) new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e){
        res.getDropOffLocation().setIsLocationAvailable(isLocationAvailable:false);
        res.getDropOffLocation().updateLocation();
        c.setIsReserved(isReserved:true);
        c.updateCar();
        jf.dispose();
        ReservationPage resPage=new ReservationPage(res,cus);
    }
});
```

(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



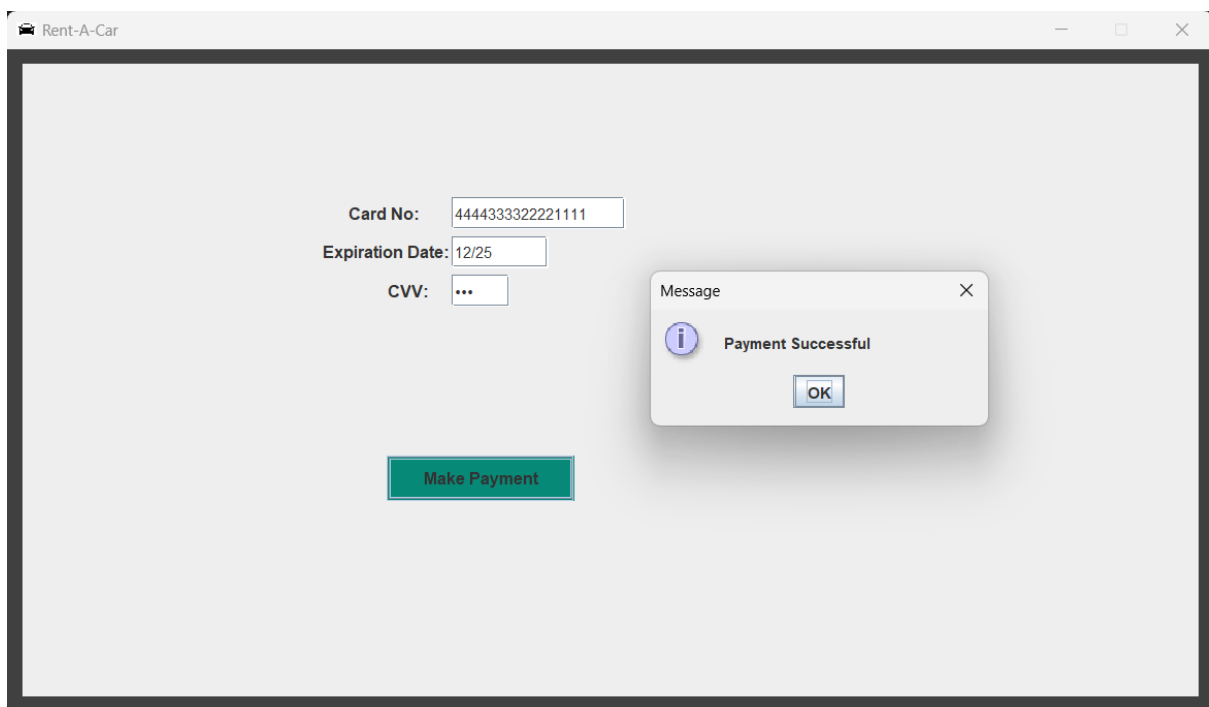
The screenshot shows a web browser window titled "Rent-A-Car". The main content area is light gray and contains the following reservation details:

- Car Brand-Model: Skoda-Scala
- Pickup Date: 30-05-2023
- Pickup Time: 16:54:05
- Drop-Off Date: 31-05-2023
- Drop-Off Time: 16:54:05
- Pickup Location: Konak, Peltek Mh. 596/4.Sk
- Drop-Off Location: Menemen, Kantar Mh. 153/5.Sk

Below the details is a green button labeled "Make Reservation".

(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.



The screenshot shows a web browser window titled "Rent-A-Car". The main content area is light gray and contains the following payment information:

- Card No: 4444333322221111
- Expiration Date: 12/25
- CVV: ...

Below the information is a green button labeled "Make Payment".

A modal dialog box titled "Message" is open, displaying a blue information icon and the text "Payment Successful". There is an "OK" button at the bottom of the dialog.

(Figure 23. screenshot payment page)

```

PaymentPage(Reservation res, Customer cus) {
    super(jf, jp);

    createLabel(text: "Card No:", x: 265, y: 115, width: 115, height: 25, jp);
    JTextField cardNoField = createField(x: 345, y: 115, width: 135, height: 25, jp);
    createLabel(text: "Expiration Date: ", x: 245, y: 145, width: 125, height: 25, jp);
    JTextField expDateField = createField(x: 345, y: 145, width: 75, height: 25, jp);
    createLabel(text: "CVV:", x: 295, y: 175, width: 85, height: 25, jp);
    JTextField cvvField = createPasswordField(x: 345, y: 175, width: 45, height: 25, jp);

    JButton makePay = createButton(text: "Make Payment", x: 295, y: 315, width: 145, height: 35, jp);
    Color c1 = new Color(r: 6, g: 137, b: 119);
    makePay.setBackground(c1);
    makePay.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (cardNoField.getText() != null && cvvField.getText() != null && expDateField.getText() != null) {
                Payment newPay = new Payment(cardNoField.getText(), expDateField.getText(), cvvField.getText());

                if (newPay.isAllFieldsValid(newPay)) {
                    JOptionPane.showMessageDialog(parentComponent: null, message: "Payment Successful");
                    jf.dispose();
                    newPay.setpaymentDate();
                    newPay.setpaymentTime();
                    newPay.setCustId(cus.getIdNumber());
                    newPay.setPaymentAmount(res.getCar().getDailyRentalRate());

                    res.setPayment(newPay);
                    res.setCustId(cus.getIdNumber());

                    newPay.addToDocument();
                    res.addToDocument(res);

                    LaunchPage launchP = new LaunchPage();
                }
            }
        }
    });
}

```

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

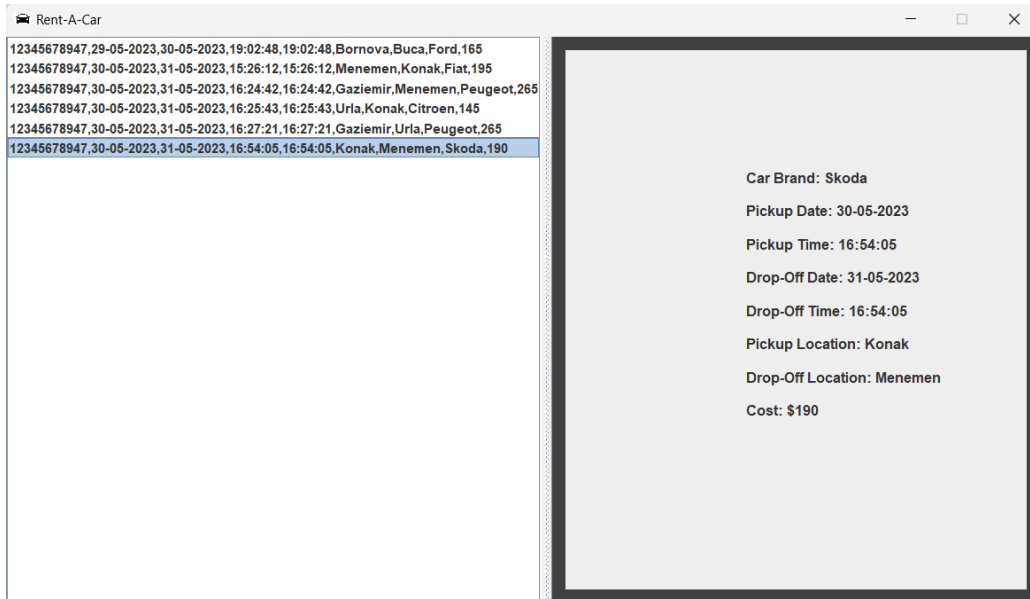
If the customer clicks on your reservation history button, she/he will be directed to the reservation history page. And here she/he can see all reservations that she/he did in her/his account.

```

JButton history = createButton(text: "Your Reservation History", x: 350, y: 235, width: 200, height: 45, jp);
history.setBackground(c1);
history.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        jf.dispose();
        ResHisPage resPage = new ResHisPage(cus.getIdNumber());
    }
});

```

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



(Figure 26. screenshot reservation history page)

```

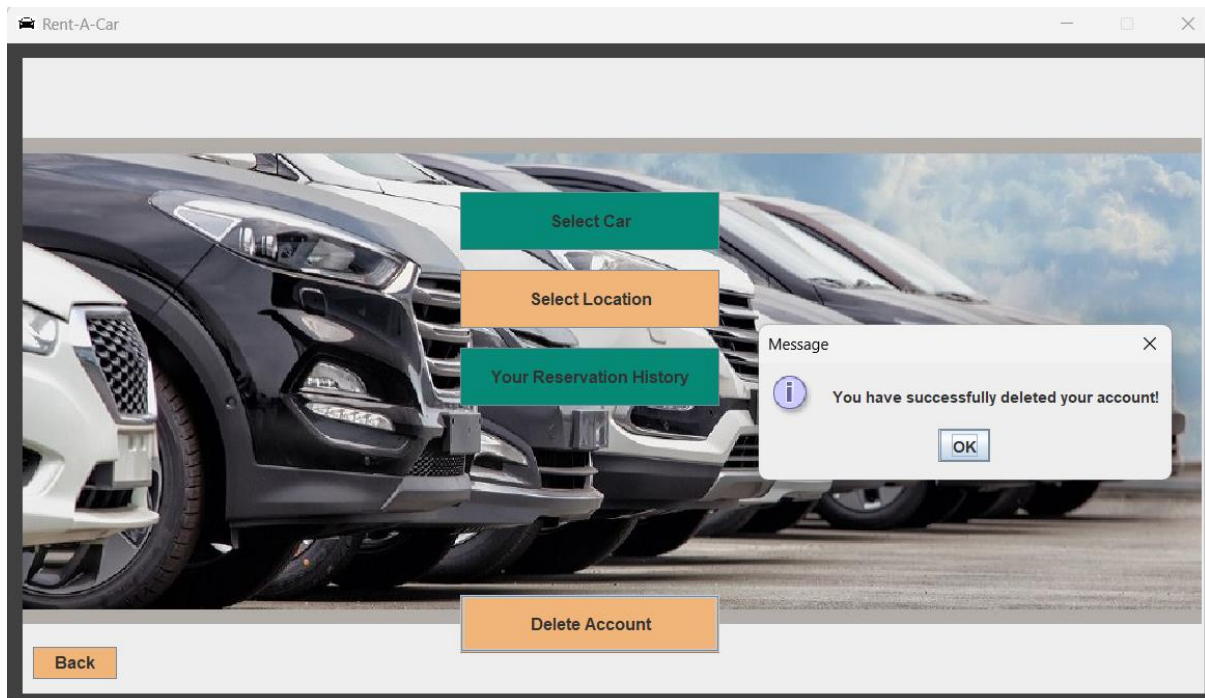
class ResHisPage extends Page{
    static JPanel jp = new JPanel();
    static JFrame jf = new JFrame(title:"Rent-A-Car");
    JList<String> list = new JList<>();
    DefaultListModel<String> model = new DefaultListModel<>();
    JSplitPane sp = new JSplitPane();
    JLabel car =createLabel(text:"", x:175, y:115, width:150, height:25,jp);
    JLabel pickDate =createLabel(text:"", x:175, y:145, width:250, height:25,jp);
    JLabel pickTime =createLabel(text:"", x:175, y:175, width:250, height:25,jp);
    JLabel dropDate =createLabel(text:"", x:175, y:205, width:250, height:25,jp);
    JLabel dropTime =createLabel(text:"", x:175, y:235, width:250, height:25,jp);
    JLabel pickLoc =createLabel(text:"", x:175, y:265, width:380, height:25,jp);
    JLabel dropLoc =createLabel(text:"", x:175, y:295, width:380, height:25,jp);
    JLabel cost =createLabel(text:"", x:175, y:325, width:380, height:25,jp);
    JLabel id =createLabel(text:"", x:175, y:355, width:380, height:25,jp);
    ResHisPage(String custId){
        super(jf,jp);
        jf.add(sp);
        list.setModel(model);
        sp.setLeftComponent(new JScrollPane(list));
        sp.setRightComponent(jp);
        Reservation.getReservationsHistory(model,custId);
        list.getSelectionModel().addListSelectionListener(e ->{
            String[] line = list.getSelectedValue().split(regex:"");
            car.setText("Car Brand: "+line[7]);
            pickDate.setText("Pickup Date: "+line[1]);
            pickTime.setText("Pickup Time: "+line[3]);
            dropDate.setText("Drop-Off Date: "+line[2]);
            dropTime.setText("Drop-Off Time: "+line[4]);
            pickLoc.setText("Pickup Location: "+line[5]);
            dropLoc.setText("Drop-Off Location: "+line[6]);
            cost.setText("Cost: $" +line[8]);
            if(custId ==null){
                id.setText("Customer's Id: "+line[0]);
            }
        });

        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jf.setResizable(resizable:false);
        jf.setVisible(b:true);
    }
}

```

(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)

```
 JButton delAcc = createButton(text:"Delete Account", x:350, y:425, width:200, height:45, jp);
 delAcc.setBackground(c11);
 delAcc.addActionListener(new ActionListener() {
     @Override
     public void actionPerformed(ActionEvent e){
         cus.deleteAccount();
         JOptionPane.showMessageDialog(parentComponent:null, message:"You have successfully deleted your account!");
         jf.dispose();
         LaunchPage launchPage = new LaunchPage();
     }
 });
```

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

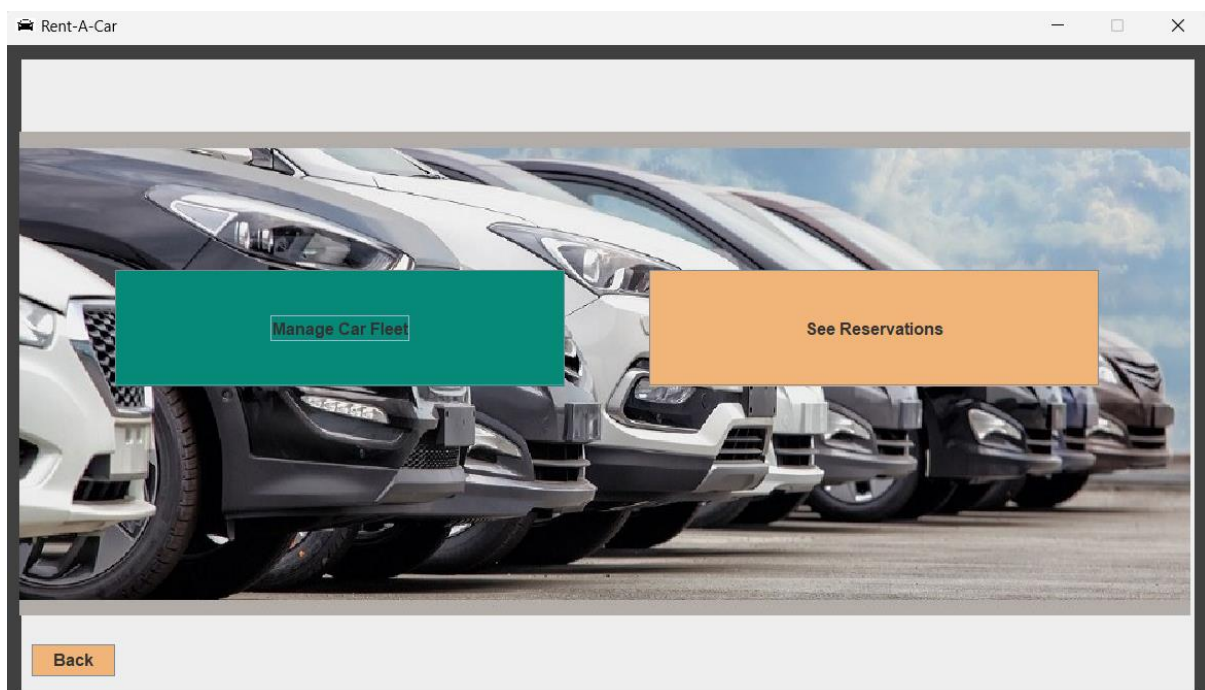

```

public void deleteAccount(){
    try {
        File cusFile = new File(pathname:"accounts.csv");
        File tempFile = new File(pathname:"temp.csv");
        FileWriter cusWriter = new FileWriter(tempFile);
        Scanner cusScanner = new Scanner(cusFile);
        cusWriter.write(cusScanner.nextLine()+"\n");
        while(cusScanner.hasNext()){
            String cusLine = cusScanner.nextLine();
            String[] cusAttr = cusLine.split(regex:",");
            if(!(this.idNumber.equalsIgnoreCase(cusAttr[3]))){
                cusWriter.write(cusLine+"\n");
            }
        }
        cusScanner.close();
        cusWriter.close();
        cusFile.delete();
        tempFile.renameTo(cusFile);
    }
    catch (IOException e) {
        e.printStackTrace();
    }
}

```

(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)

```

//When user presses Log In button
logIn.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e){
        String mail = mailField.getText();
        String password = passwordText.getText();
        if(mail.equals(anObject:"admin@rentacar.com")&& password.equals(anObject:"12345")){
            jf.dispose();
            AdminPage adminPage = new AdminPage();
        }
    }
});

```

(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.

```

AdminPage(){
    super(jf, jp);

    JButton carFleet = createButton(text:"Manage Car Fleet", x:85, y:175, width:350, height:90, jp);
    Color c1 = new Color(r:6, g:137, b:119);
    carFleet.setBackground(c1);

    carFleet.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e){
            jf.dispose();
            FleetPage fleetPage = new FleetPage();
        }
    });
}

```

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

Rent-A-Car

objectpack.Car@3e4a6d78
objectpack.Car@4c1f36ad
objectpack.Car@4b17477b
objectpack.Car@191db8e3
objectpack.Car@74d13715
objectpack.Car@414edd7c

Please type in with the given format(CSV).

Selected Car: Fiat,500,2019,Pink,4.9,195,34MT975,Menemen

Add Car

Remove Car

Update Car

(Figure 34. screenshot of fleet page)

```

FleetPage(){
    super(jf, jp);
    jf.add(sp);
    list.setModel(model);
    sp.setLeftComponent(new JScrollPane(list));
    sp.setRightComponent(jp);
    Car.getList(model,1,null);

    JTextField addField = createField(x:170, y:170, width:380, height:30, jp);
    JButton addCar = createButton(text:"Add Car", x:280, y:210, width:140, height:30, jp);
    Color c1 = new Color(r:66, g:137, b:119);
    addCar.setBackground(c1);
    addCar.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e){
            jf.dispose();
            String[] carAttr = addField.getText().split(regex);
            Car c = new Car(carAttr[0], carAttr[1], carAttr[2], carAttr[3], carAttr[4], carAttr[5], carAttr[6], isReserved:false, carAttr[7]);
            c.addToCSV();
        }
    });
    JButton removeCar = createButton(text:"Remove Car", x:280, y:270, width:140, height:30, jp);
    Color c11 = new Color(r:242, g:181, b:121);
    removeCar.setBackground(c11);
    JTextField updateField = createField(x:170, y:360, width:380, height:30, jp);
    JButton updateCar = createButton(text:"Update Car", x:280, y:400, width:140, height:30, jp);
    updateCar.setBackground(c1);
    list.getSelectionModel().addListSelectionListener(e ->{
        Car c = list.getSelectedValue();
        carName.setText("Selected Car: "+c.getBrandName()+" "+c.getModelName()+" "+c.getYear()+" "+c.getColor()+" "+c.getFuelConsumption()+" "+c.getDailyRentalRate()+" "+c.getLicensePlate());
        updateCar.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e){
                jf.dispose();
                String[] carAttr = updateField.getText().split(regex);
                Car fieldCar = new Car(carAttr[0], carAttr[1], carAttr[2], carAttr[3], carAttr[4], carAttr[5], carAttr[6], isReserved:false, carAttr[7]);
                fieldCar.updateCar();
            }
        });
    });
}

```

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

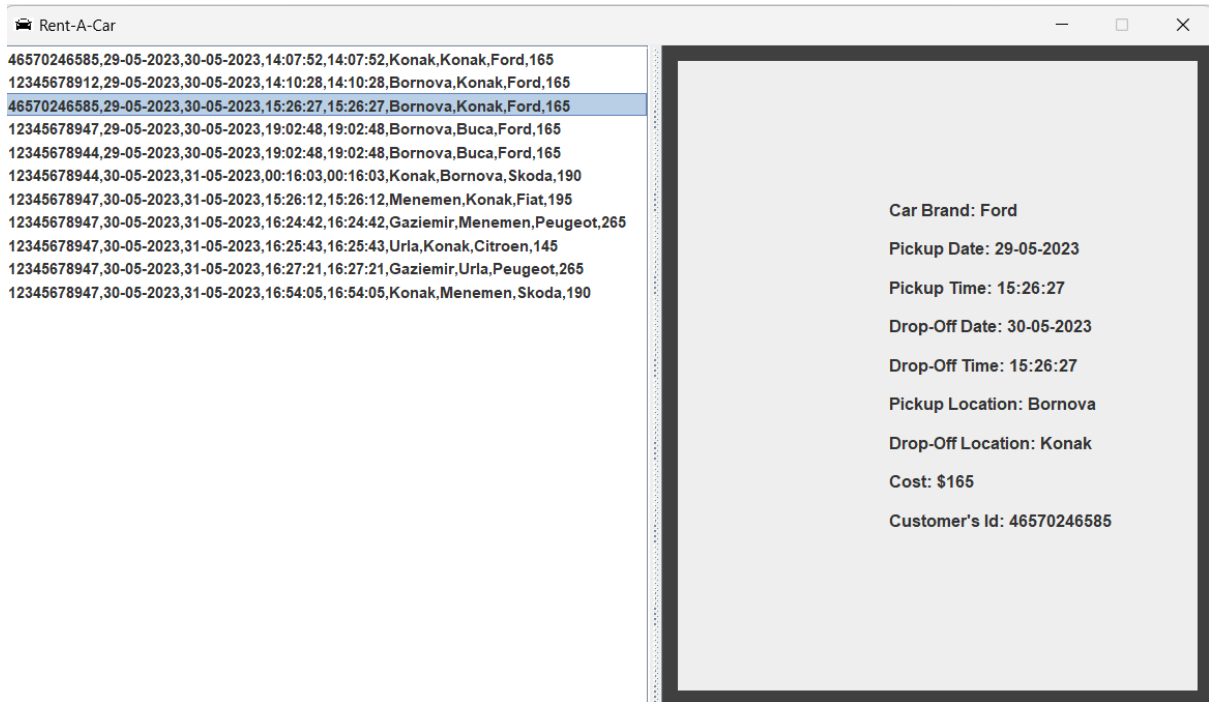
If he/she clicks on the see reservations button, he is directed to the reservation history page. Here, the admin can see the all reservations.

```

JButton reservationPanel = createButton(text:"See Reservations",x:500 , y:175, width:350, height:90, jp);
Color c11 = new Color(r:242, g:181, b:121);
reservationPanel.setBackground(c11);
reservationPanel.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e){
        jf.dispose();
        ResHisPage resPage = new ResHisPage(custId:null);
    }
});

```

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



(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.