Car rental system

Group Assignment

Module Code	**:**	CT038-3-2-OODJ – Object Oriented Development with Java
Intake Code	**:**	APD2F2209CS(CYB)
Lecturer Name	**:**	Dr. USMAN HASHMI
Hand in Date	**:**	Week 13 – the 16th of December 2022

Student ID	**Student Name**
TP066168	MOHAMED KHAIRY MOHAMED ABDELRAOUF
TP064361	ABDELRAHMAN MOURAD ABDELSATTAR RAMADAN

Table of Contents

1 ABSTRACT

The "Car Rental System" was developed to automate the manual system using computerised equipment and cutting-edge software. This was done to fulfil their requirements so that their valuable data and information could be stored for a more extended period and could be manipulated more easily. The essential software is easily accessible and does not need a steep learning curve.

This application can manage the system's generic operations, such as accepting or rejecting client reservations made using the same system and adding new automobiles to the fleet of vehicles available for reservation through the system.

2 ASSUMPTIONS

- Customers can change their account password and username.
- The car rental system application operates 24 hours a day.
- The admin has permissions to everything while respecting clients' privacy.
- Customers cannot apply to be an admin.
- Admin adds new cars every week.
- There are no limits for the customer to rent cars as long as he has enough money.
- The admin is the only one authorized to add new accounts, whether it’s admin or client accounts.
- Admin can view every single transaction in the application.
- Clients can view their profiles and car rental status.
- The admin is the only one who has the authority to accept or reject the rental process.
- Both admin and customer will login in the same page.
- Customer can’t I register to the program until he is 18 years old or above.

3 ACKNOWLEDGEMENT

Mr. Usman Hashmi, our lecturer, was always inspiring, and we wish to convey our gratitude to him first and foremost. He inspired us to complete this task without hesitation and urged us to think imaginatively. He provided us with all the assistance and teaching tools required to complete this assignment. Additionally, we are grateful to each member of this group. This project was successful due to the efforts made by each individual. Because we were always there to encourage one another, we remained united until the end.

We are grateful to all of our companions who extended their moral support, but most of all, we are grateful to God for being with us and allowing us to accomplish this endeavor.

4 INTRODUCTION

4.1 PROJECT OUTLINE

This project aims to provide a framework for car rentals that will enable users, both consumers and administrators, to interact with the system by logging in with their respective credentials. The framework will enable the administrator to add vehicles, manage appointments, and carry out all of the other vital tasks anticipated in the vehicle rental framework.

Similarly, users will be able to register by giving some basic information, logging in to the system, and making bookings utilising the data about the cars that are now accessible to them.

4.2 MOTIVATION

The present manual automobile rental systems are inefficient and difficult for consumers to use since they do not allow them to monitor the status of their vehicles from the comfort of their homes. Because of this, there is a need for a centralised system that will make it simpler to use and more usable overall, ultimately resulting in increased productivity.

4.3 PROBLEM IDENTIFICATION

Since the companies' physical method of managing customers and resource management is out of date, results in many errors, and requires much work, the organizations must implement a centralized system.

4.4 PROJECT GOALS

The system must be able to perform the following functions:

- Customer registration in the system.
- Allows logging in separately for Administrator and Clients.
- Vehicle data management.
- Validate reservations.
    - Acceptance of the reservation
    - Rejection of reservation
- Save the log in a .txt file

5 REQUIREMENT ANALYSIS

In this part, we will explore the needs of the system, as well as its design and analysis, using the relevant software models.

5.1 SYSTEM REQUIREMENTS

The requirement analysis phase of the system development cycle is significant since it is the phase that provides us with all of the detailed specifications of the system, which are crucial for the building of the system, as well as information about the behaviour of the system. The information acquired throughout collecting system requirements is used to construct the system.

This information also serves as a source for the later phases of the system development life cycle. The needs of the system may be broken down into the two categories that are listed below:

- Functional Requirements
- Non-Functional Requirements
- Extra-features

5.2 FUNCTIONAL REQUIREMENTS

The system's behaviour, functionalities, and objectives are all precisely defined in Functional Requirements. Use cases will demonstrate the behaviour requirements, which are functions that describe the system's behaviour. The following table explains the significance and description of functional requirements:

5.3 NON-FUNCTIONAL REQUIREMENTS

In computer systems, quality qualities are often referred to as non-functional requirements. Consequently, it runs counter to the FR because it gives us information on operations rather than behaviour or functionalities. The system's design includes definitions of non-FR, which both help us achieve our quality goals and improve the system's operation. These are requirements that have nothing to do with functionality:

5.4 EXTRA-FEATURES

They are additional features to improve the system's overall performance and broaden the scope of its functional goals. The relevance of the functional needs and a description of them are listed below in the table.

Table 1: Functional Requirements

Num	FR	IMPORTANCE	DESCRIPTION
1	User_Login	Essential	Enables the user to log into the program
2	Admin_Options	Essential	Options available to the Admen
3	Add_Account	Essential	Admin can add new accounts.
4	Add_Cars	Essential	The admin might edit car data.
5	Manage_Bookings	Essential	Booking acceptance by admin
6	Report	Essential	Displays customer bookings
7	Book_Appointment	Essential	Enables customers to book automobiles

Table 2: Non-Functional Requirements

Num	NON-FR	IMPORTANCE	DESCRIPTION
1	Welcome	Essential	System performance should be rapid.
2	User Friendly	Essential	Easy-to-use interfaces are simple.
3	Application Upkeep	Essential	Thoroughly maintain and document.
4	Application Scalability	Essential	Flexible and easily evolveable systems.

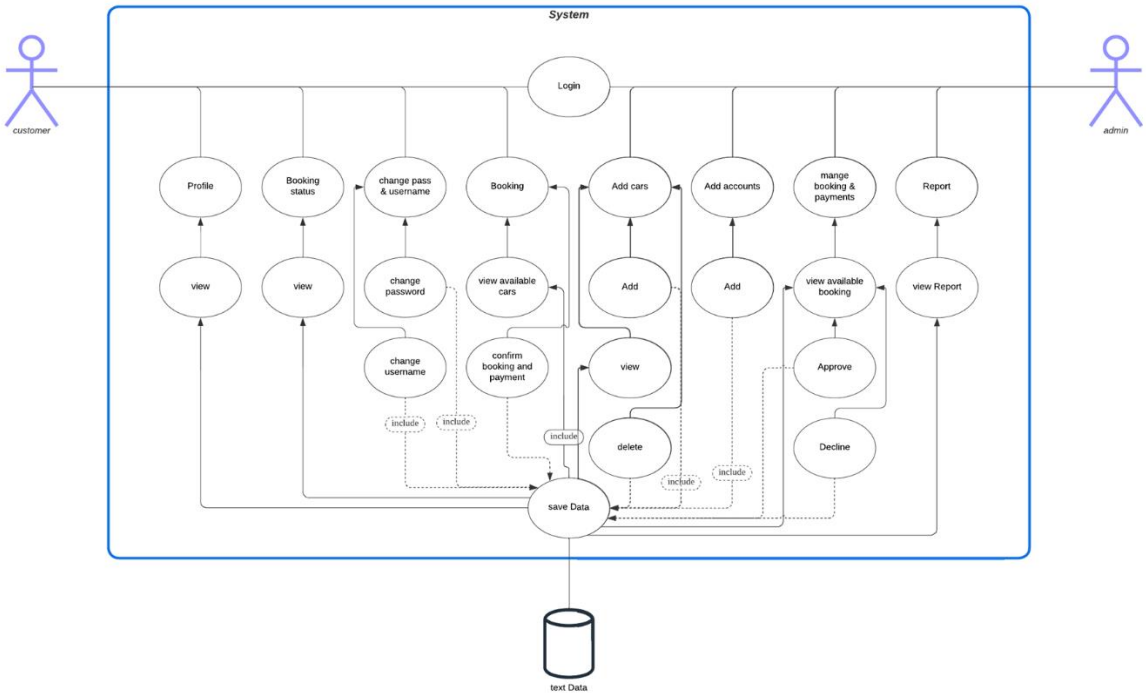
Table 3: Extra-features

Num	XF	IMPORTANCE	DESCRIPTION
1	Welcome	Additional	Welcome user
2	Status_booking	Additional	View booking status
3	Secret	Additional	You have to discover it yourself
4	Profile	Additional	View client information
5	CU_Options	Additional	Software client activities
6	Change_Pass_User	Additional	Allows consumers to reset their password

6 REQUIREMENTS ANALYSIS

6.1 SYSTEM USE CASE

This section discusses a potential application of the car rental system. The system's use can be illustrated by examining a set of use scenarios. The three primary players in our setup are the System Administrator, the Customer, and the Data Store. Every user has their designated function and level of access to the system.



6.2 USE CASE DESCRIPTION

Use Case	Login
Brief Description	Allows users and admin to sign-in to the system via username and password.
Actors	Admin/ Customer
Pre-Condition	The admin is responsible for adding accounts to clients or even to another admin
Main Flow	Provide Username and Password
Alternative Flow	Display login error if the login credentials are wrong
Post-Condition	Customer or Admin has successfully logged in to the system.

Use Case	Profile
Brief Description	That allows the user to display his personal data
Actors	Customer
Pre-Condition	Exhibit the client's specific information
Main Flow	• After registering, the customer will enter the program and after moving to the customer list. • She/He will have to click on the profile until the display option appears.
Alternative Flow	Go back to the home page
Post-Condition	Display the option to view customer information

Use Case	View (Profile)
Brief Description	Responsible for displaying customer data
Actors	Customer
Pre-Condition	You can't get here without signing up as a customer.
Main Flow	• After accessing the profile page, • The customer must click on the View button to display his personal information.
Alternative Flow	Go back to the home page
Post-Condition	View customer information

Use Case	**Booking status**
Brief Description	To inform the customer whether his reservation has been accepted or not
Actors	Customer
Pre-Condition	Log in as a customer
Main Flow	• After logging in and accessing the customer's menu. • The customer must go to the booking status menu, where he will find options for displaying reservation data.
Alternative Flow	Go back to the home page
Post-Condition	View customer information

Use Case	**View (Booking status)**
Brief Description	Extract text from the txt.file and display it
Actors	Customer
Pre-Condition	VIEW button lets consumer see booking data and details
Main Flow	• After accessing the profile page, • To see his booking information, the customer must click the View button.
Alternative Flow	Go back to the home page
Post-Condition	The information appears on the page in the designated locations.

Use Case	**Change pass & username**
Brief Description	It is the list that authorizes the customer to change his personal data
Actors	Customer
Pre-Condition	Change the username and password
Main Flow	• After accessing the customer's menu and logging in. • The customer will have to choose the page to change the password and username. • Where there will be spaces to insert the new password and username
Alternative Flow	Go back to the home page
Post-Condition	Type the new username and password

Use Case	**Booking**
Brief Description	A list that allows the user to book (rent) a car.
Actors	Customer
Pre-Condition	A list users can consult to rent a car with or without a driver.
Main Flow	• After accessing the customer's menu and logging in. • The customer will have to choose the page to booking a cord. • Then the booking page will appear with the available options.
Alternative Flow	Go back to the home page
Post-Condition	Enter booking data

Use Case	**Change password**
Brief Description	It is the button that authorizes changing the password data in the text file.
Actors	Customer
Pre-Condition	Here is the button that lets you edit the text file containing your password.
Main Flow	• After accessing the page, change the username and password. • Then fill in the new password in the space provided for it. • The customer will then have to click on the Change Password button, • In which he will be able to change the password in the text file.
Alternative Flow	Error, Invalid password / Go back to the home page
Post-Condition	Password changed successfully

Use Case	**Change username**
Brief Description	It is the button that authorizes changing the username data in the text file.
Actors	Customer
Pre-Condition	Here is the button that lets you edit the text file containing your username.
Main Flow	• Subsequent to getting to the page "change the username and secret word". • Next, enter the new username in the box provided. • After clicking "Change Username," the client can change the text file's username.
Alternative Flow	Error, Invalid username / Go back to the home page
Post-Condition	Username changed successfully

Use Case	View available cars
Brief Description	It is the button that displays a list of cars that are currently available.
Actors	Customer
Pre-Condition	Show cars currently available
Main Flow	• The customer will need to select a vehicle from the available options once he or she enters the car reservation section. • However, you must press the "Show Available Cars" button to display available automobiles.
Alternative Flow	Error, please choose a car / Go back to the home page
Post-Condition	Car has been selected successfully

Use Case	Confirm booking and payment
Brief Description	This button is for confirming the reservation process after selecting the car
Actors	Customer
Pre-Condition	This button confirms car reservations after selection.
Main Flow	• After the car show is over, select the available vehicle and determine whether or not it comes with a driver. • To confirm the reservation process, we will instruct the customer to press this button.
Alternative Flow	Error, The process has not been completed / Go back to the home page
Post-Condition	The car has been booked successfully

Use Case	Add cars
Brief Description	It's the button that takes you to the page for adding cars.
Actors	Admin
Pre-Condition	Add cars and define their types
Main Flow	• After successfully logging in, the administrator will be presented with a list of admins from which he can choose to add a car. • He will be added to the list of cars to add automatically.
Alternative Flow	Go back to the home page
Post-Condition	Go to the add cars page

Use Case	Add (Add cars)
Brief Description	It is the button responsible for adding the car to the list of available cars
Actors	Admin
Pre-Condition	This is the option that will include the vehicle in the list of options.
Main Flow	• After moving to the "car data management" page. • Admins will have to enter the car information that will be added as a car. • Then they will press the Add button until they confirm the addition of this new car.
Alternative Flow	Error, missing data / Go back to the home page
Post-Condition	Car has been added successfully

Use Case	View (Add cars)
Brief Description	View the cars that have been added
Actors	Admin
Pre-Condition	It is in charge of displaying the text file's data in the designated locations on the page.
Main Flow	• After moving to the car data management file. • When the admin adds a car, he can view the car that has been added by clicking on the view button. • Then you can view the added cars.
Alternative Flow	Error, no cars have been added / Go back to the home page
Post-Condition	The added car has been successfully displayed

Use Case	Delete (Add cars)
Brief Description	Delete a car from the auto text file
Actors	Admin
Pre-Condition	Remove a vehicle from the text file's cars
Main Flow	• After moving to the car data management file. • So that the admin can delete any existing car. • He must first view the available cars by clicking on the View button.
Alternative Flow	Error, No car has been selected / Go back to the home page
Post-Condition	The selected car has been deleted

Use Case	**Add accounts**
Brief Description	It is the one that authorizes the admin to move to the list of adding accounts
Actors	Admin
Pre-Condition	It gives the administrator ability to add accounts.
Main Flow	The administrator can add accounts after logging in and looking through the admin list.
Alternative Flow	Go back to the home page
Post-Condition	Go to the page of adding accounts

Use Case	**Add (Add accounts)**
Brief Description	Add the new account data to the text file
Actors	Admin
Pre-Condition	Adding the new account information to the program's database
Main Flow	• After going to the Add Accounts page. • The admin must add all the required data and then press the Add button so that the new data is saved in the text file
Alternative Flow	Error, missing data / Go back to the home page
Post-Condition	The account has been added successfully

Use Case	**Mange booking & payments**
Brief Description	Transfer the admin to the list of booking
Actors	Admin
Pre-Condition	Move the administration over to the booking list.
Main Flow	• After the admin logs in. • Going through the admin list, he can choose to “manage booking & payments”. • Which will take him to the reservations display page.
Alternative Flow	Go back to the home page
Post-Condition	Display mange booking & payments

Use Case	View available booking
Brief Description	It is the button that allows the admin to view booking requests
Actors	Admin
Pre-Condition	Which allows the admin to see the reservations from customers
Main Flow	• After the admin navigates to the Manage Bookings And Payment menu. • The admin must click on the view available booking button.
Alternative Flow	Go back to the home page
Post-Condition	View available booking successfully displayed

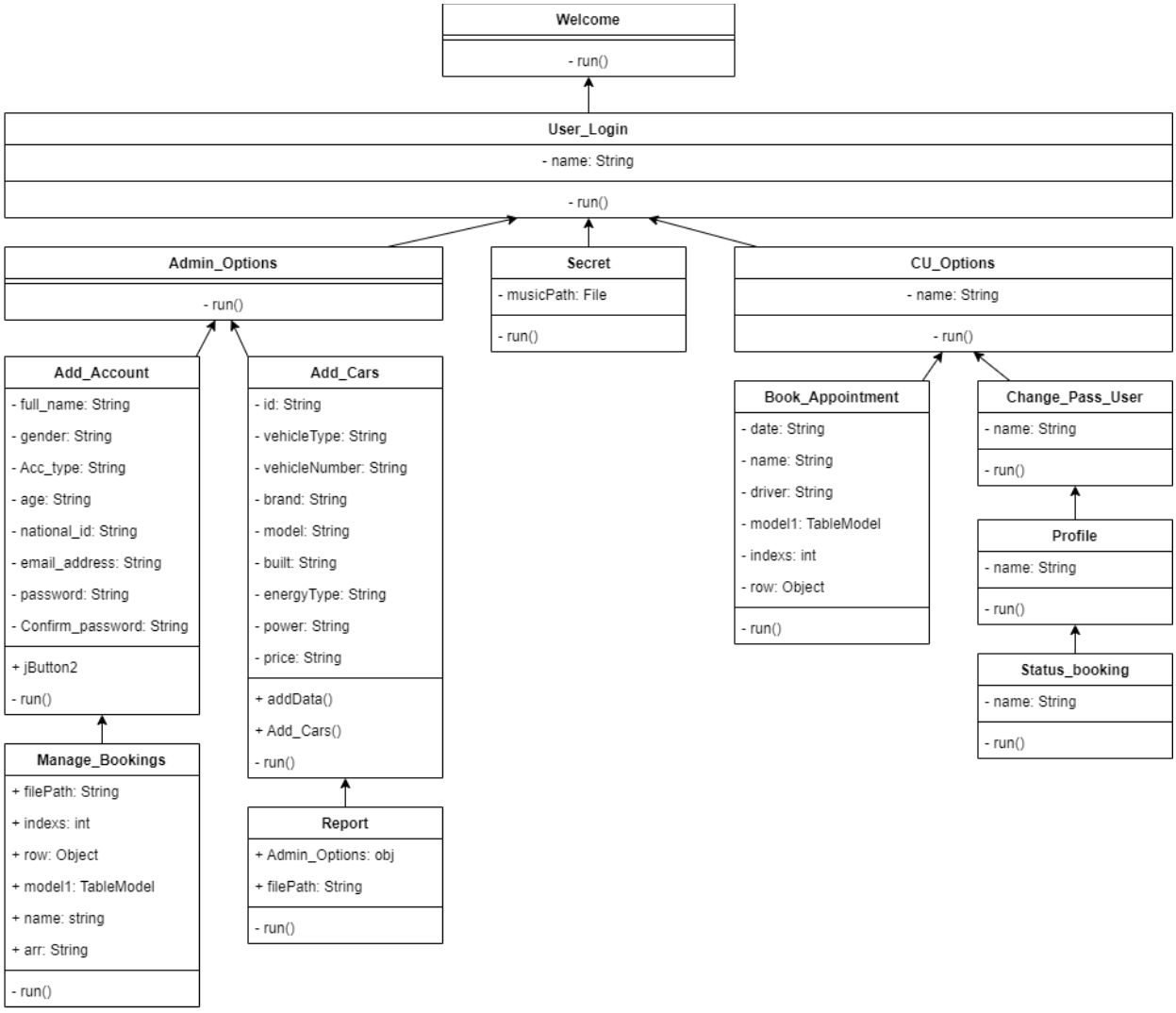
Use Case	Approve (mange booking & payments)
Brief Description	Car reservation accepted
Actors	Admin
Pre-Condition	The Use Case is in charge of moving the text file transaction from waiting to accepting.
Main Flow	• After going to the "mange booking & payments" page, and then clicking on the available reservations view. • The admin can, when authorized, choose between accepting or rejecting the reservation. • This Use Case is responsible for acceptance.
Alternative Flow	Go back to the home page
Post-Condition	The case has been accepted successfully

Use Case	Decline (mange booking & payments)
Brief Description	Car reservation refused
Actors	Admin
Pre-Condition	The use case is responsible for moving the text file transaction from waiting to rejecting.
Main Flow	• After clicking on the view of available reservations after going to the "mange booking & payments" page. • With permission, the administrator can either accept or reject the reservation. • Acceptance is dependent on this Use Case.
Alternative Flow	Go back to the home page
Post-Condition	The case has been declined successfully

Use Case	Report
Brief Description	Go to the report view page
Actors	Admin
Pre-Condition	The admin is taken to the report display page by this use case.
Main Flow	After the admin logs in, going directly to the admin menu, he/she can choose "Report".
Alternative Flow	Go back to the home page
Post-Condition	Display the report page

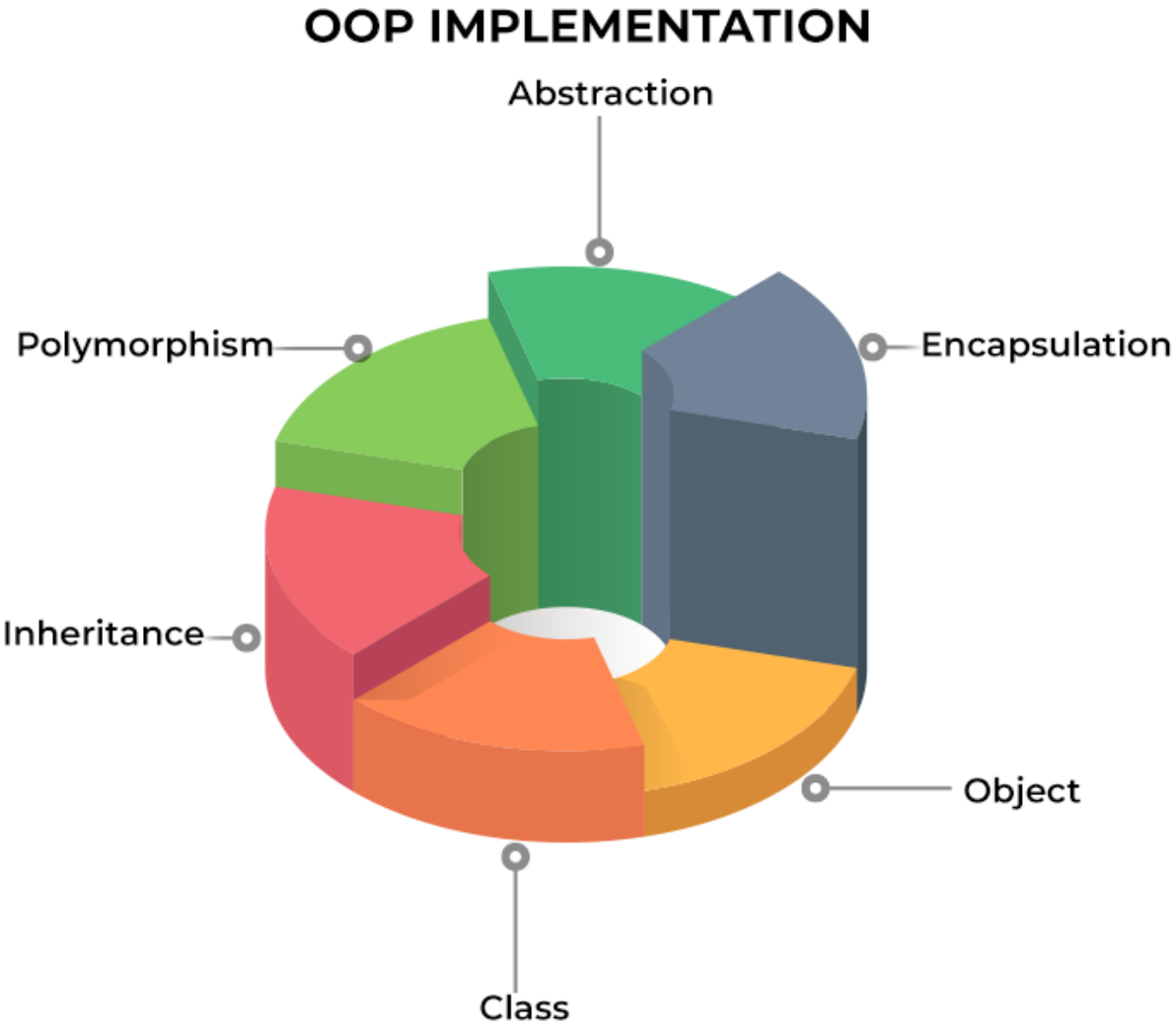
Use Case	View Report
Brief Description	Responsible for fetching data from a text file and displaying it in a table
Actors	Admin
Pre-Condition	Retrieves and displays text file data in a table.
Main Flow	• After the admin moves to the report page. • It can view the data of the transactions made by the customers by clicking on the display button.
Alternative Flow	Error, no available data / Go back to the home page
Post-Condition	Data have been viewed successfully.

6.3 CLASS DIAGRAM



7 SYSTEM IMPLEMENTATION

7.1 OBJECT ORIENTED PROGRAMMING



Object-oriented programming (OOP) is a programming paradigm in which data, and objects, take precedence over functions and logic throughout software development. An item is a particular data field with its history and behaviour.

In contrast to procedural programming, object-oriented programming (OOP) places a premium on the objects themselves. This programming style is ideal for large, complex, and regularly updated or maintained projects. OOP, for instance, can be used in software for the

simulation of production systems and is thus applicable across a wide range of applications, from mobile apps to design and manufacturing programmes.

Non-functional requirements are qualitative qualities in computer systems. It contradicts the FR since it describes procedures, not behaviour or functionality. Object-oriented programmes are useful for collaborative programming, which entails subdividing larger jobs. OOP provides code efficiency, scalability, and reuse.

Despite being influenced by its forerunners, Java was not created to share source code with any other language. Therefore, the Java team was able to start their planning process from scratch. One outcome was an organisation system that made objects more accessible and tidier. Java borrows extensively from many of the most influential object-software environments of the previous few decades, allowing it to find a happy medium between the purist's "everything is an object" paradigm and the pragmatist's "keep out of my way" model.

As simple types like integers continue to be high-performance nonobjects in Java, its object architecture is intuitive and easy to adapt.tem's design includes definitions of non-FR, which both help us achieve our quality goals and improve the system's operation. These are requirements that have nothing to do with functionality:

7.1.1 Objects

When you make a class, a new data type is born. You can use this type to declare instances of that type. It is important to note that obtaining class objects requires a two-stage process. As a first step, declare a variable of class type. This variable does not specify what an object is. It is a variable that can reference a specific object.

Second, you must replace that variable concerning a physical instance of the object. The new administrator will allow you to do this. The new operator dynamically allocates storage space for an object and then returns a reference to it (at run time). This reference generally corresponds to the memory address of the object allocated by the new.

This reference will subsequently be stored in the variable. So, Java needs to use dynamic allocation for all class objects.

```
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new User_Login().setVisible( b: true);
        }
    });
}
```

7.1.2 Inheritance

The concept of inheritance is a crucial component of OOP (Object-Oriented Programming). It is the technique in Java that allows one class to inherit the characteristics (fields and methods) of another class. Inheritance can only occur between classes that extend the same superclass. In Java, "inheritance" refers to the process of building new classes by copying and adapting existing ones.

A class that inherits from another class is able to make use of the other class's pre-existing methods and properties. You also have the ability to add additional fields and methods to the class you are currently working with.The system was constructed so that it would fulfil the requirements mentioned earlier successfully.

Due to the limitation of not using a database, which would have made things more professional and accessible, we created and developed this system to fulfil those functional and non-functional needs, as mentioned earlier. In doing so, we were able to meet all of the requirements. More features can also be added, but since we are running short of time, we cannot accommodate them.

```
public class User_Login extends javax.swing.JFrame {
String name;
```

7.1.2.1 Why do we need it?

The code that is written in the Superclass can be reused in all of the subclasses because it is shared throughout all of them. The code from the parent class can be used directly in the child classes.

Method Overriding: In order to succeed in Method Overriding, Inheritance is your only option. Run-time polymorphism is one of the goals that can be accomplished by Java using this method.

Inheritance is the mechanism that allows us to realise the concept of abstraction, which states that we do not have to disclose all of the information. The user is only presented with the functionality while using abstraction.

7.1.2.2 Important terminologies

Class: A class is a set of objects that all have the same features, behaviours, and properties. Class is not a thing that exists in the real world. It is nothing more than a format, a blueprint, or a prototype that may be used to fashion new things.

The class from which other classes receive their characteristics is referred to as the parent class. A superclass is the class from which other classes inherit their characteristics (or a **base class or a parent class**).

A class that inherits the characteristics of another class is referred to as a "subclass," which is synonymous with "child class" (or **a derived class, extended class, or child class**). In addition to the attributes and methods provided by the superclass, the subclass is able to provide its own unique fields and methods.

The idea of "**reusability**" is one that is supported by inheritance. This means that if we want to create a new class but there is already a class that contains some of the code that we want, we can derive our new class from the existing class. In other words, we can reuse the code that is already written. By acting in this manner, we are reusing the fields and methods of the class that already exists.

7.1.3 Encapsulation

Java's encapsulation is a robust mechanism for storing the class's data members and data methods in a unified fashion. This is done by creating a protected field that can only be accessed by other objects of the same class.

The Java concept of encapsulation refers to the bundling together of data (variables) and the code that manipulates that data (methods). Through the process of encapsulation, the variables of a class are protected from being accessed by other classes and are instead made available only to the methods of the class.

7.1.3.1 What is Encapsulation in Java?

Encapsulation in Java refers to integrating data (variables) and code (methods) into a single unit. In encapsulation, a class's variables are hidden from other classes and can only be accessed by the methods of the class in which they are found.

Java's encapsulation mechanism is an object-oriented means of enclosing a user-defined class's data members and data methods. Declaring this class as private is essential.

Java encapsulation is a notion of object-oriented programming that defines how information and the methods that operate on it should be bundled together.

It is often used for the purpose of keeping private information hidden. This technique protects some characteristics from outsiders while letting existing class members access them using public getter and setter methods. If you use these techniques, you may control which attributes are accessible for reading or updating and ensure that any changes to an attribute's value have been validated.

Encapsulation offers the crucial concealing data characteristic for safeguarding user data. If you're using an APM solution like Retrace to catch bugs, it's a good idea to follow OOP best practises like encapsulation.

Do you want to become a professional Java Developer by expanding your knowledge of the Java programming language and passing the associated certification exams? Then you should

look at our Java training and certification curriculum, which has been compiled by some of the most seasoned working professionals in the field.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    name=jTextField1.getText();
    String password=jPasswordField1.getText().toString();


    try {
        Scanner in = new Scanner(new File( pathname: "Customer_Data.txt"));
        while (in.hasNextLine()){
            String s = in.nextLine();
            String[] sArray = s.split( regex: "/");

            if (name.equals(sArray[0]) && password.equals(sArray[5]) && sArray[6].equals( anObject: "CU")){
                JOptionPane.showMessageDialog( parentComponent: null, message: "Login Successful", title: "Success", messageType: JOptionPane.INFORMATION_MESSAGE);
                dispose();
                CU_Options obj=new CU_Options( Uname: name);

                obj.setVisible( b: true);
                break;
            }
            else if (name.equals(sArray[0]) && password.equals(sArray[5]) && sArray[6].equals( anObject: "AD")){
                JOptionPane.showMessageDialog( parentComponent: null, message: "Login Successful", title: "Success", messageType: JOptionPane.INFORMATION_MESSAGE);
                dispose();
                Admin_Options obj=new Admin_Options();
                obj.setVisible( b: true);
                break;
            }
             else if (name.equals( anObject: "CYBERPUNK") && password.equals( anObject: "CYBERPUNK") ){
                JOptionPane.showMessageDialog( parentComponent: null, message: "Login Successful", title: "Success", messageType: JOptionPane.INFORMATION_MESSAGE);
                dispose();
                Secret obj=new Secret();
                obj.setVisible( b: true);
                break;
            }

        }

        in.close();


    }
    catch (FileNotFoundException e) {
        JOptionPane.showMessageDialog( parentComponent: null,
                message: "User Database Not Found", title: "Error",
                messageType: JOptionPane.ERROR_MESSAGE);
    }
```

7.1.4 Classes

A class is a blueprint for an object. A class, which can be thought of as a notion, is represented by an object. Having a class established is the first step in creating any object. Let us say you are developing software and intend to add a human component. You must be able to characterise the person and motivate them to take action.

The "person" class would serve as a template for what a human being is and what they are capable of doing; to work with a person in your code, you would need to create a new object. The person class makes objects with a "person" data type. You can now describe this person and have it take action based on your description.

Classes are pretty beneficial for learning how to programme. Consider the case where a hundred persons would be more beneficial than one. Instead of painstakingly drawing each one from scratch, you can use a standard individual class to generate a hundred objects of the 'individual' type with identical visual architecture but unique names and details.

```
public class Car_Rental_System {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
      Welcome abc=new Welcome();
       abc.setVisible( b: true);
    }

}
```

7.1.5 General Form of Class

When a class is defined, its exact appearance and characteristics are made known for the first time. You do this by detailing its contents and the logic behind its processing. While some elementary classes may consist of code, most real-world classes have both code and data. As we will see, the code for a class defines the access point for its data. A class is declared with the class keyword. The classes employed thus far represent only a fragment of the complete set. Classes can, and often do, become far more involved.

When a class is defined, its exact appearance and characteristics are made known for the first time. You do this by detailing its contents and the logic behind its processing. While some elementary classes may consist of code, most real-world classes have both code and data. As we will see, the code for a class defines the access point for its data.

A class is declared with the class keyword. The classes employed thus far represent only a fragment of the complete set. Classes can, and often do, become far more involved.The information stored in a class is called "instance variables." The actual programming is done in the methods. Its members comprise a class, including all its associated variables and methods.

Class-specific methods manage and provide access to the vast majority of instance variables held by classes. Thus, methods determine which applications can use data from a particular class.

For this reason, "instance variables" refer to variables duplicated for each class instance or object in the class. Accordingly, information about one thing is not interchangeable with

information about another. We will return to this topic soon, but know that it is crucial to your understanding. In our experience, all methods take the form of Main().

Most methods, however, will not be marked as either public or static. Take note that there is no overarching class specification for the primary () method. Java classes do not necessarily have a primary function. If you are using that class as the foundation for your software, you only need to provide a single instance.

7.1.6 Polymorphism

In polymorphism, an object can change into other shapes. In OOP, polymorphism is most frequently employed when a reference to a parent class is used to refer to an object of a child class. Java objects are termed polymorphic if they pass more than one IS-A test. Because any object in Java may pass the IS-A test for both its own type and the Object class, all Java objects are inherently polymorphic.

Keep in mind that a reference variable is the sole way to get at an object. There is just one sort of reference variable. Once a reference variable has been declared, its type cannot be modified. If the variable is not marked as final, it can be reassigned to different objects. The reference variable's type specifies which of the object's methods it can call.

The value of a reference variable can be any object of the defined type or any subtype of that type. Declaring a reference variable to be of a class or interface is possible.

7.1.7 Abstraction

Abstraction from data is the quality that allows for the presentation of only the most pertinent information to the end user. The user is not shown the irrelevant or unnecessary measurements. Example: people tend to think of a car as a whole instead of its parts. The term "data abstraction" can also refer to the process of selecting only the relevant features of an object and discarding the rest.

Differentiating one object from another of the same type is made possible through its properties and behaviours, which also aid in classifying and grouping them. Think about a man behind the wheel in the real world. The man is unaware of the inner mechanism of the car or the actual implementation of the accelerator, brakes, etc. in the vehicle; he only knows that pressing the accelerators will raise the speed of a car and that applying the brakes would stop the car.

To put it simply, this is abstraction. Abstraction is performed in Java through the use of interfaces and abstract classes. Through the use of interfaces, we can accomplish full abstraction.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
```

7.1.7.1 When to use

At times, we may need to build a superclass that only describes the structure of an abstraction, rather than provide a full implementation of every method. Occasionally, we'd like to make a superclass that just specifies a generalisation form that all its subclasses should use, leaving the specifics to the subclasses themselves.

Let's take the ubiquitous "shape" example, which may be found in everything from a CAD programme to a virtual world. Each shape has its own colour, size, and so on, and "shape" is the primary type. Shapes like circles, squares, triangles, and so on can be derived (inherited) from this basic shape and may display unique traits and behaviours.

Certain shapes, for instance, can be inverted. When attempting to determine the area of a shape, for example, you may encounter some unexpected behaviour. The similarities and distinctions between the shapes are reflected in the type hierarchy.

8 IMPLEMENTATION/ USER MANUAL

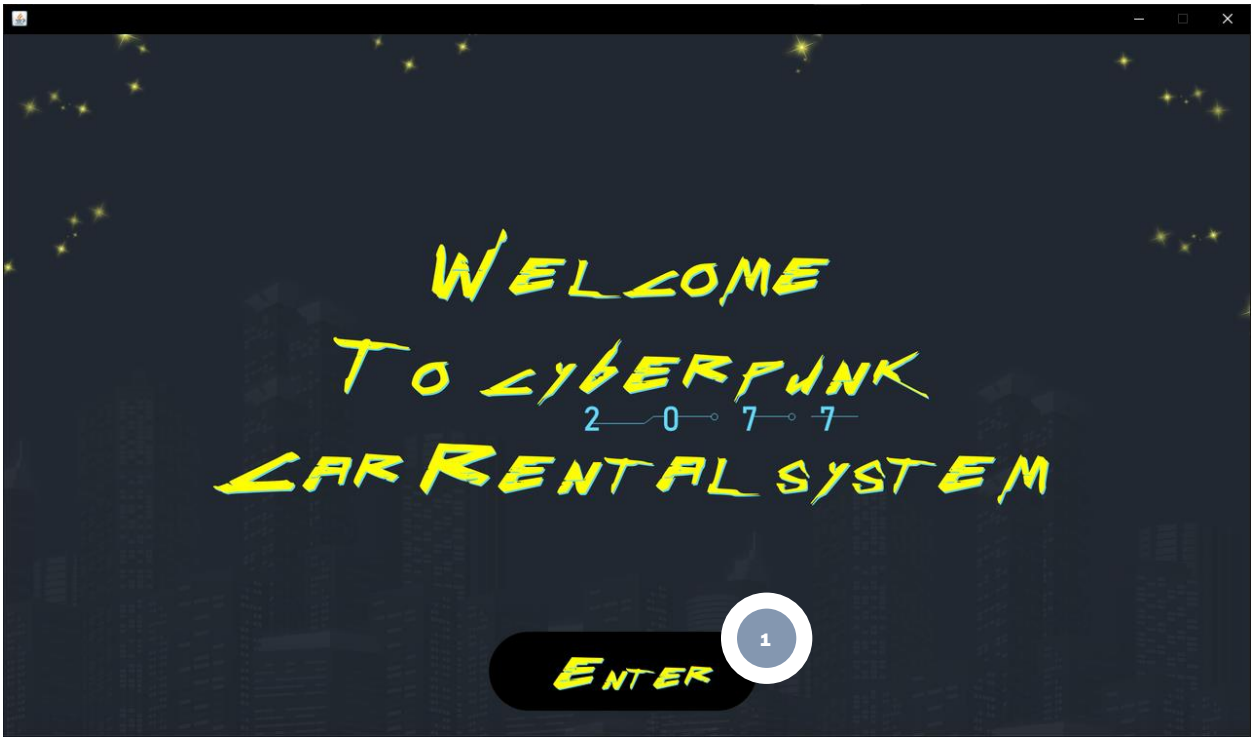
User guides are documents that show the end user how to use a system, product, or service correctly and effectively. This document is also known as an instruction manual or user guide. Data on activities, principles and rules, investigating manuals, usefulness, and more can be generally tracked down in such distributions.

Client manuals ordinarily remember investigating exhortation and definite directions for how to appropriately work the item. Without expecting readers to read the entire document, you can anticipate that a table of contents and index will assist them in quickly locating the information they require.

A quick start guide ought to be included at the beginning of the manual in order to assist users in becoming productive immediately. It can come on paper structure, be posted on the web, or be a mix of the two.

8.1 BOTH ADMIN AND CUSTOMERS

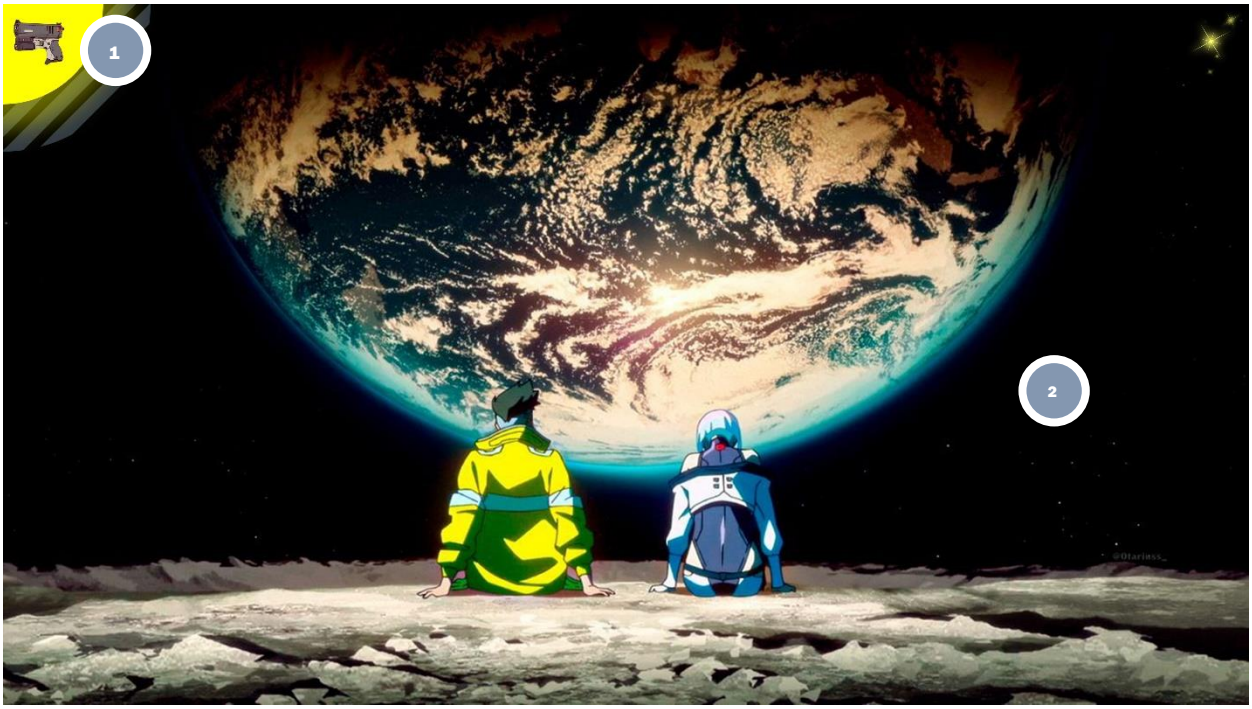
Step0



Whether the user is an admin or a customer, the welcome page will first appear, and then he or she will need to click the Enter button.

Num	Name	Palette	Function
1	Enter	Button	Go to the login page

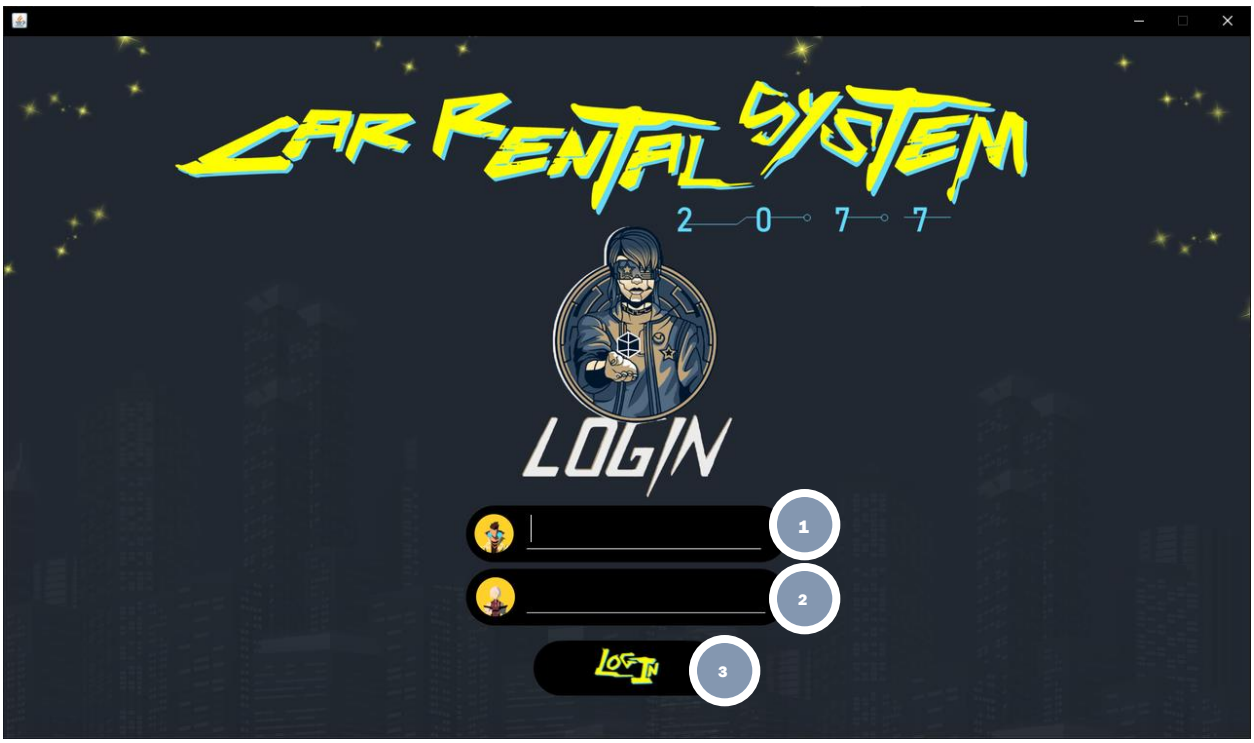
Step1



It is a page with no specific purpose other than to display a song from the cartoon series that is inspired by the program's theme you can access it by typing CYBERPUNK in username and CYBERPUNK in password.

Num	Name	Palette	Function
1	jButton5	Button	Return to the previous page
2	jButton1	Button	To play or pause the song

Step2



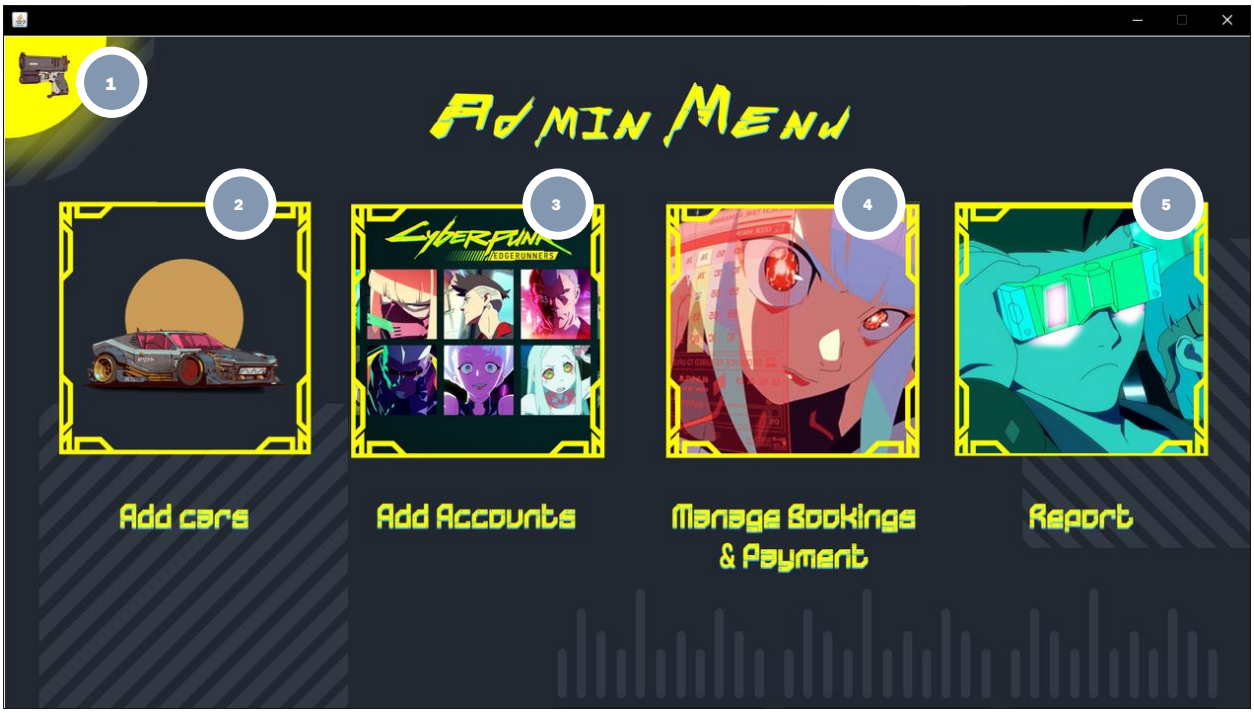
The user, whether an admin or a customer, will have to enter the username and password in order to be able to enter the program

Afterward, the user, depending on whether he is an admin or a client, will be moved to his section.

Num	Name	Palette	Function
1	jTextField1	jTextField	Write the username
2	jTextField2	jTextField	Write the password
3	Login	Button	Log in to the program

8.2 ADMIN/MANAGER USER

Step 3



The admin will automatically be taken to the admin menu section upon logging in, where he will find four options (add cars, add accounts, manage booking and payments, and report) and a "home" button at the top of the list on the left.

Num	Name	Palette	Function
1	jButton5	Button	Return to the previous page
2	jButton1	Button	Go to the add car page
3	jButton2	Button	Go to the add accounts page
4	jButton3	Button	Go to the manage bookings & payments page
5	jButton4	Button	Go to the report page

Step 4

A list for the administrator to fill in the details of the new car that needs to be added will appear starting from the "Add Cars" list.

Num	Name	Palette	Function
1	jButton5	Button	Return to the previous page
2	jTextField1	TextField	fill in the virtual ID
3	jComboBox1	ComboBox	choose the virtual type
4	jTextField2	TextField	fill in the virtual number
5	jComboBox2	ComboBox	choose the brand
6	jTextField3	TextField	fill in the model number
7	jComboBox3	ComboBox	choose the model built date
8	jComboBox4	ComboBox	choose the engine type
9	jTextField4	TextField	fill in and the power of the vehicles
10	jTextField5	TextField	fill in the price of the car
11	jButton1	Button	to add the car to the database

12	jButton2	Button	to view the cars you have in your database
13	jButton3	Button	to remove a car from database

Step 5

He can view the cars that have been added by clicking the "View" button after completing the data entry process by clicking the “add” button. He can also click the "Delete" button to delete any car from the table that is displayed.

You can also click the "home" button to go back.

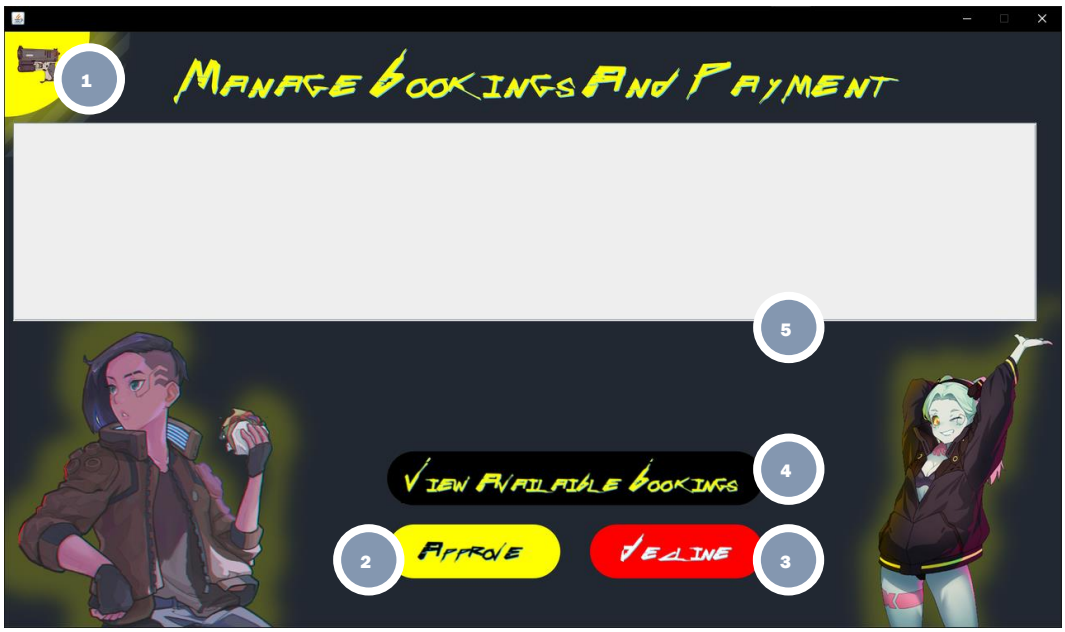
Step 6

After that, the administrator can use the admin menu to access the "Add Accounts" list. From there, he or she can add accounts, whether they are customer accounts or admin accounts, by entering the necessary information and clicking the "Add" button.

Num	Name	Palette	Function
1	jButton5	Button	Return to the previous page
2	jTextField1	TextField	enter the username
3	jTextField2	TextField	enter the e-mail address
4	jTextField3	TextField	enter the password
5	jTextField4	TextField	confirm the password
6	jTextField5	TextField	enter the national ID
7	jRadioButton1	RadioButton	choose customer
8	jRadioButton2	RadioButton	choose admin
9	jRadioButton3	RadioButton	choose male
10	jRadioButton4	RadioButton	choose female

11	jComboBox1	ComboBox	Select your age
12	jButton1	Button	Add an account

Step 7

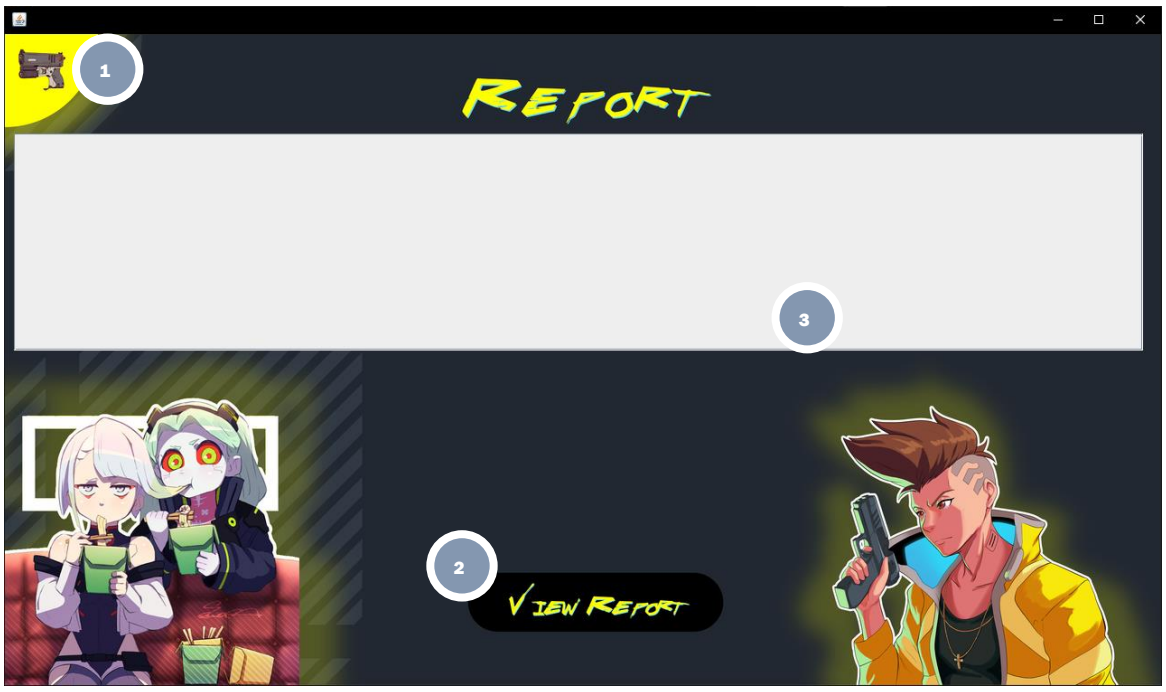


The admin menu provides access to the "Manage booking and Payment" menu. First things first, the administrator looks at the page without looking at any data. However, once the administrator clicks the button that says "View Available booking" a table will appear for him, as shown below. After selecting one of the bookings in the table, the administrator has the option to either decline or accept the reservation.

	Name	Palette	Function
1	jButton5	Button	Return to the previous page
2	jButton1	Button	view available working
3	jButton2	Button	accept or approved the booking for the customer
4	jButton3	Button	Decline or refuse the booking for the customer
5	jTable1	Table	where bookings will be viewed


Manage Bookings And Payment
Name Vehicle ID Vehicle Type Vehicle No Brand Model Build Energy Type Power Date of Booking Driver Status Paying Status Booking Status
Dazai 125 Sports 2 HONDA BMW M5 2022 Electrical 4400 CC Thu Nov 10 16:... Without Driver PAID Declined & Ref...
Pedro TREGER Sports ERGERGE HONDA ERGGE 2022 Mechanical GERGE CC Sat Dec 17 01:... Without Driver PAID Approved
Pedro TREGER Sports ERGERGE HONDA ERGGE 2022 Mechanical GERGE CC Wed Dec 28 14... With Driver PAID Waiting
View Available Bookings
Approve
Decline

Step 8



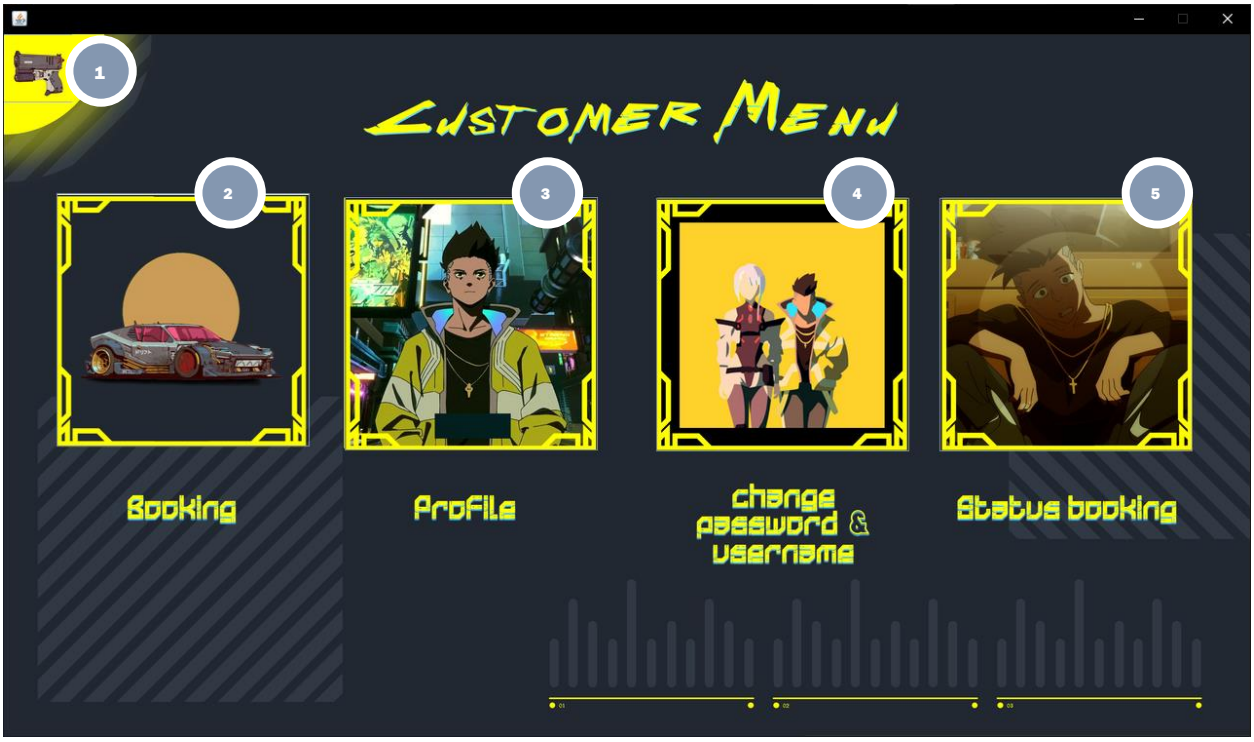
Last but not least, for the admin, he can go to the report section, where the page will appear to him or her without data, then he or she can click on the “View” button so that he or she can view the available data, which is shown in the figure below.

Num	Name	Palette	Function
1	jButton5	Button	Return to the previous page
2	jButton1	Button	to view the report
3	jTable1	Table	where the data will be viewed


REPORT
Name Vehicle ID Vehicle Type Vehicle No Brand Model Build Enerygy Type Power Date of Booking Driver Status Paying Status Booking Status
Dazai 125 Sports 2 HONDA BMW M5 2022 Electrical 4400 CC Thu Nov 10 16:... Without Driver PAID Declined & Ref...
Pedro TREGER Sports ERGERGE HONDA ERGGE 2022 Mechanical GERGE CC Sat Dec 17 01:... Without Driver PAID Approved
Pedro TREGER Sports ERGERGE HONDA ERGGE 2022 Mechanical GERGE CC Wed Dec 28 14... With Driver PAID Waiting
VIEW REPORT

8.3 CUSTOMER USER

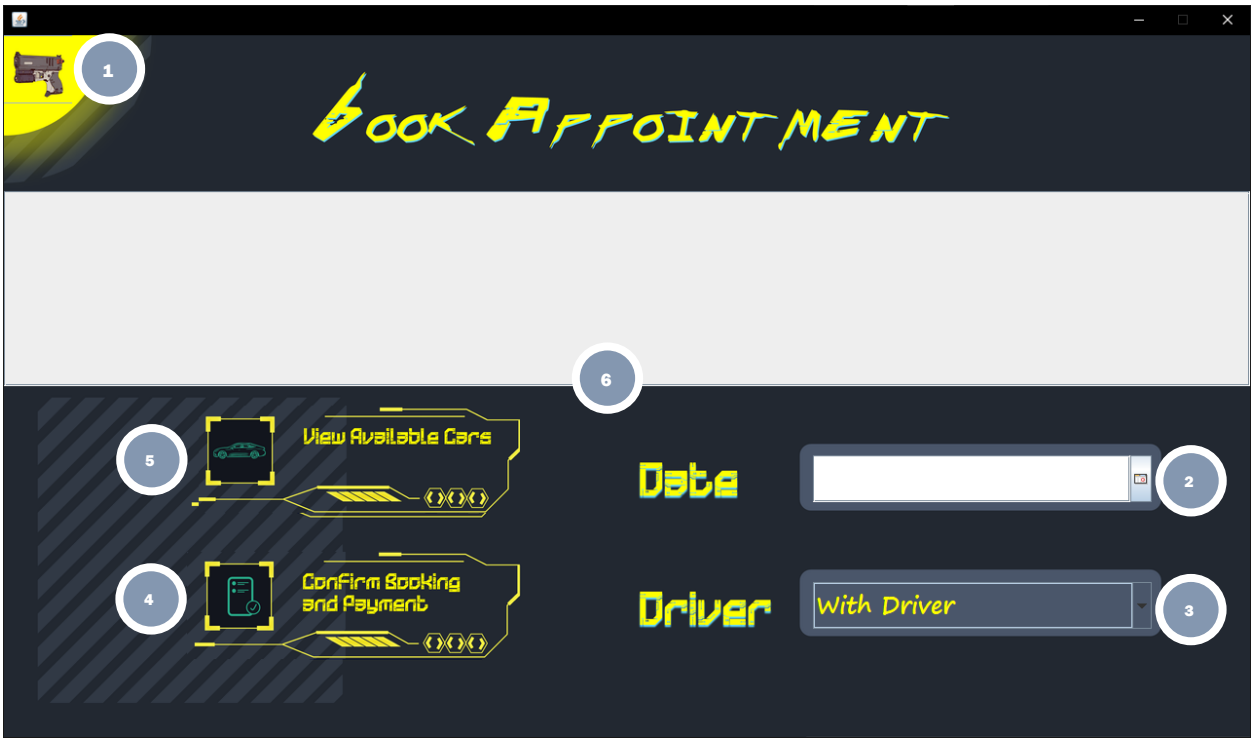
Step 3



After logging in, the customer will be automatically taken to the customer menu section, where he will find four options: booking, profile, changing password and username, and status booking. There is also a "home" button at the top of the list on the left.

Num	Name	Palette	Function
1	jButton5	Button	Return to the previous page
2	jButton1	Button	to go to the booking page
3	jButton2	Button	to go to the profile page
4	jButton3	Button	to change your password and username
5	jButton4	Button	to see your status booking

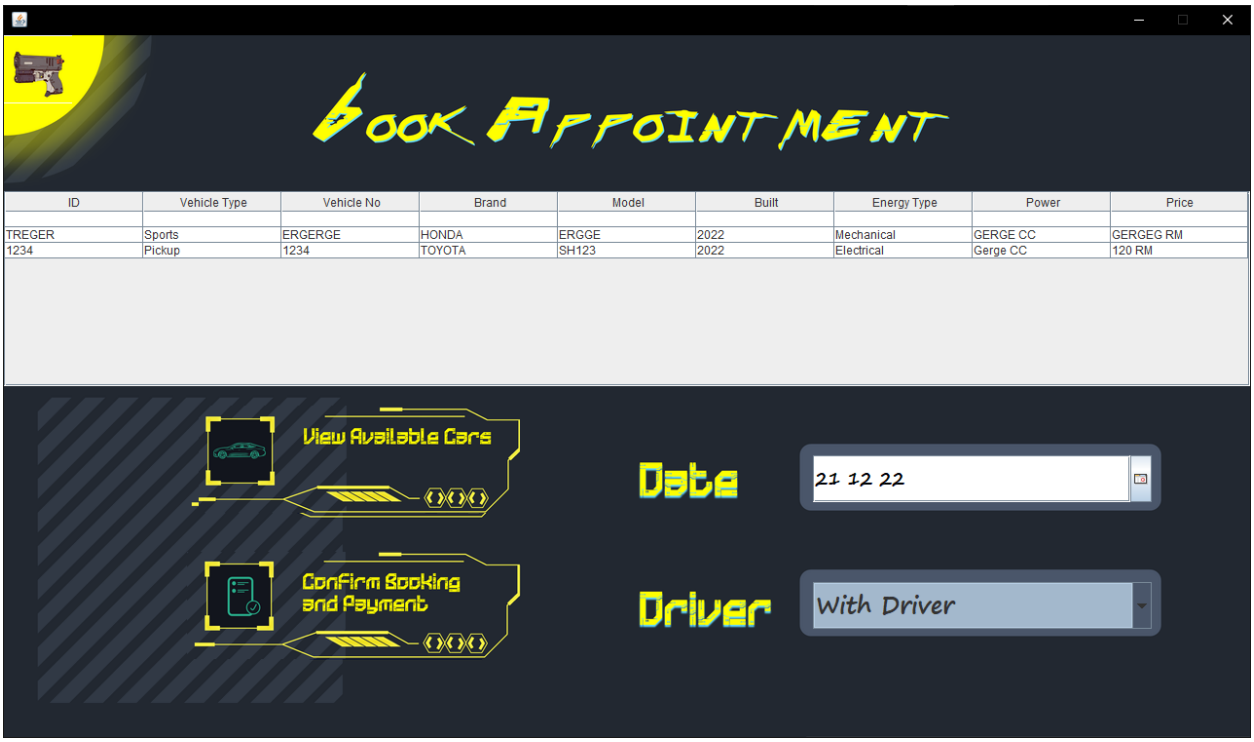
Step 4



The customer can first get to the booking page by using the customer list, and then he will move to this page, which won't have any data on it. After that, the customer will need to select "View Available Cars" in order to view the data as shown in the second step.

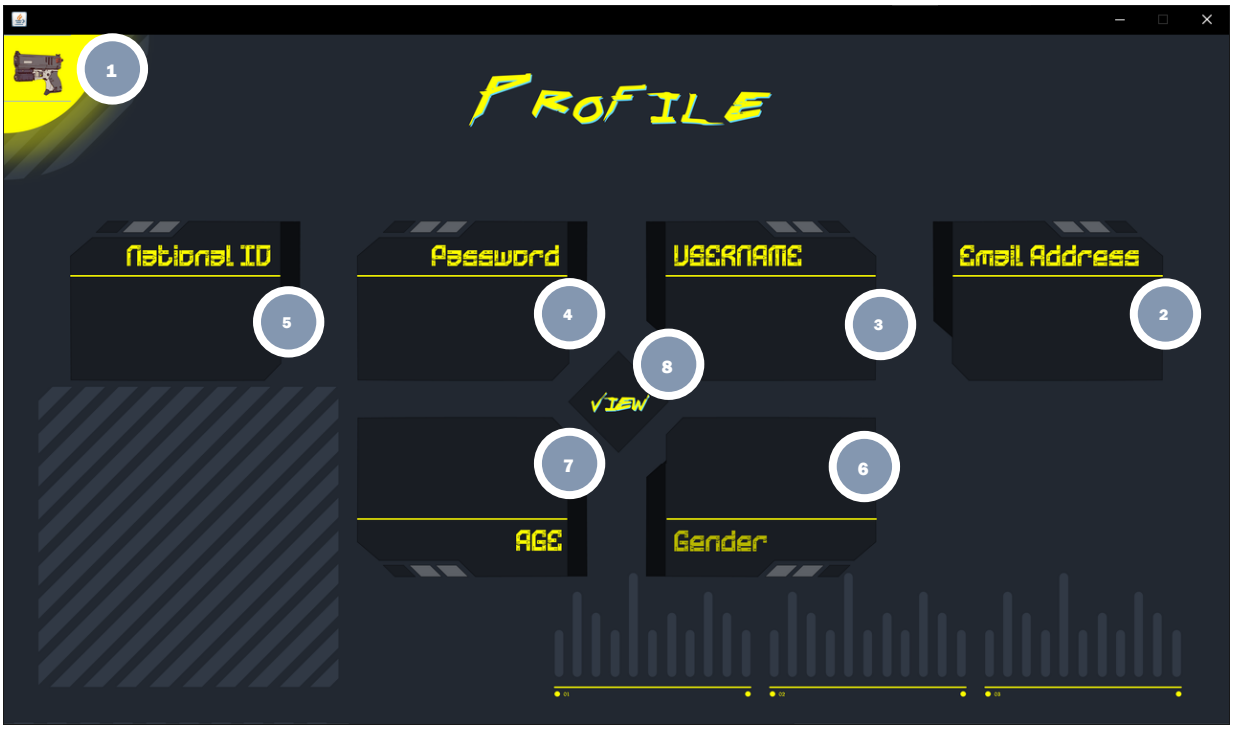
Num	Name	Palette	Function
1	jButton5	Button	Return to the previous page
2	jdatechooser2	datechooser	to choose the date
3	jComboBox1	ComboBox	to choose whether the car came with driver or not
4	jButton3	Button	to confirm booking and payment
5	jButton4	Button	to view the available car that can be booking
6	jTable1	Table	where the available car displayed

Step 5



After selecting a vehicle from the schedule, the customer must select a date and whether he/she requires a driver before clicking "Confirm Booking and Payment" to confirm the reservation.

Step 6

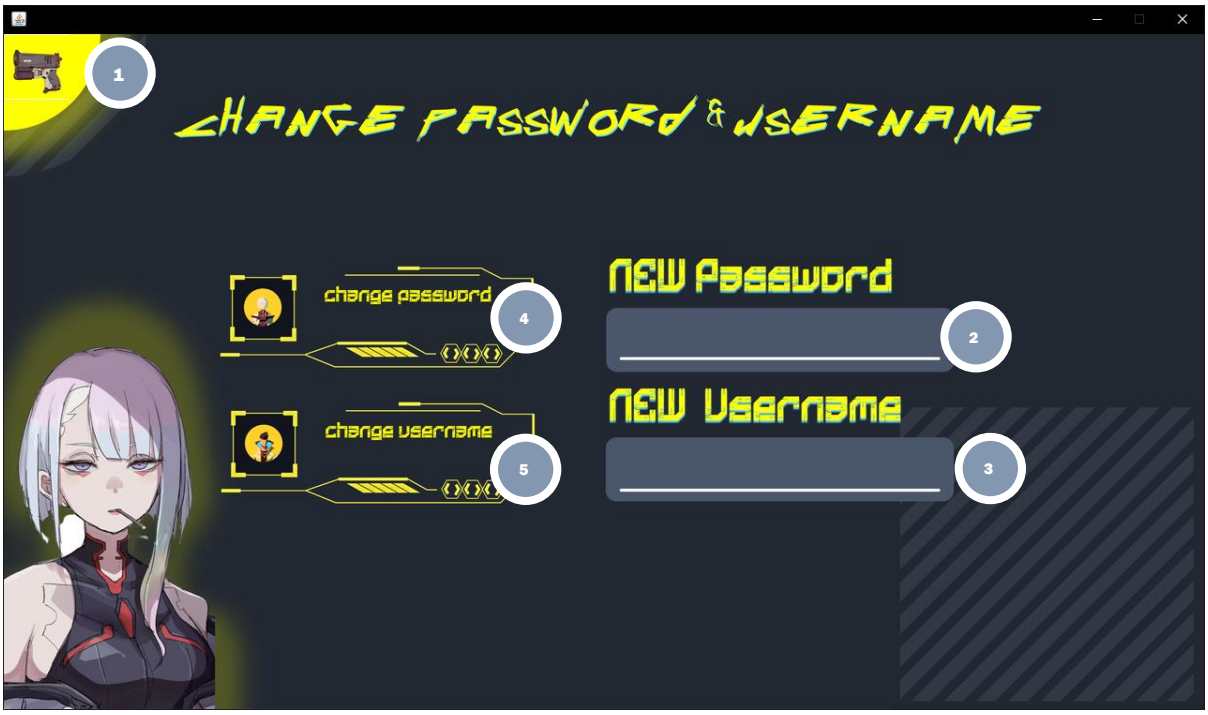


By clicking "View" on this page, the customer can view his personal information in the form shown below; This page can be accessed through the client menu, as previously mentioned.

Num	Name	Palette	Function
1	jButton5	Button	Return to the previous page
2	jLabel6	Label	where the e-mail of the customer will be displayed
3	jLabel5	Label	where the username of the customer will be displayed
4	jLabel4	Label	where the password of the customer will be displayed
5	jLabel2	Label	where the national ID of the customer will be displayed
6	jLabel3	Label	where the gender of the customer will be displayed
7	jLabel7	Label	where the age of the customer will be displayed
8	jButton1	Button	you can click here to display all customer's information

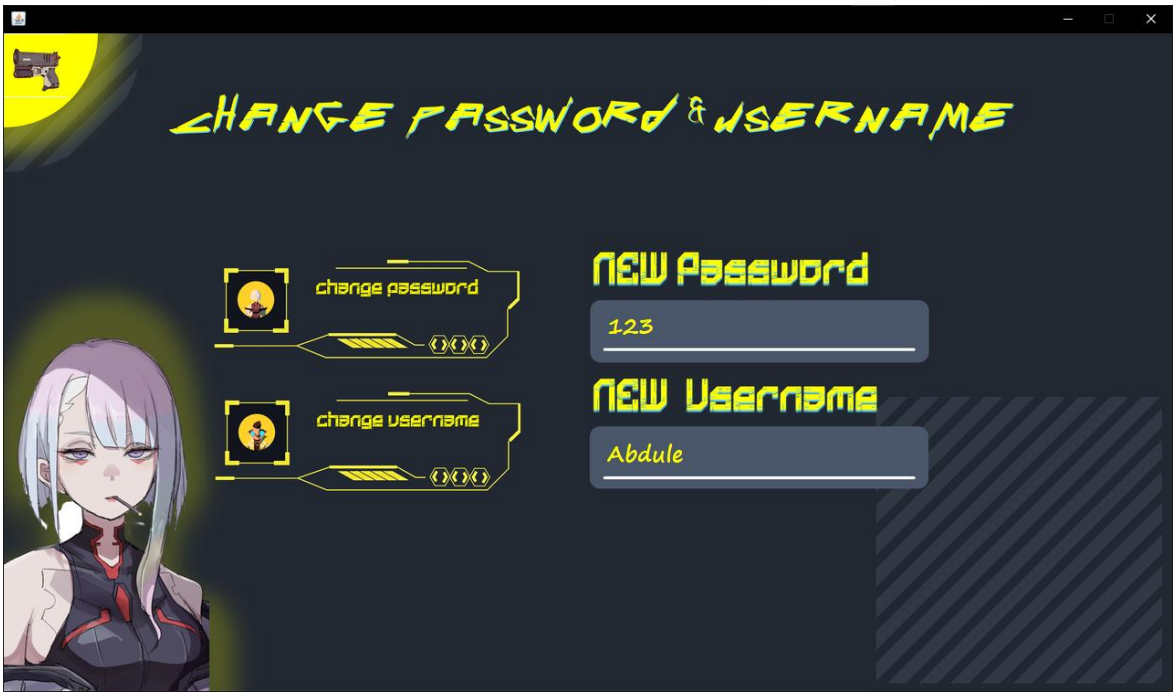

PROFILE
National ID
CU12345
Password
123
USERNAME
Abdelrahman
Email Address
Abdelrahman@gmail....
VIEW
18
AGE
male
Gender

Step 7

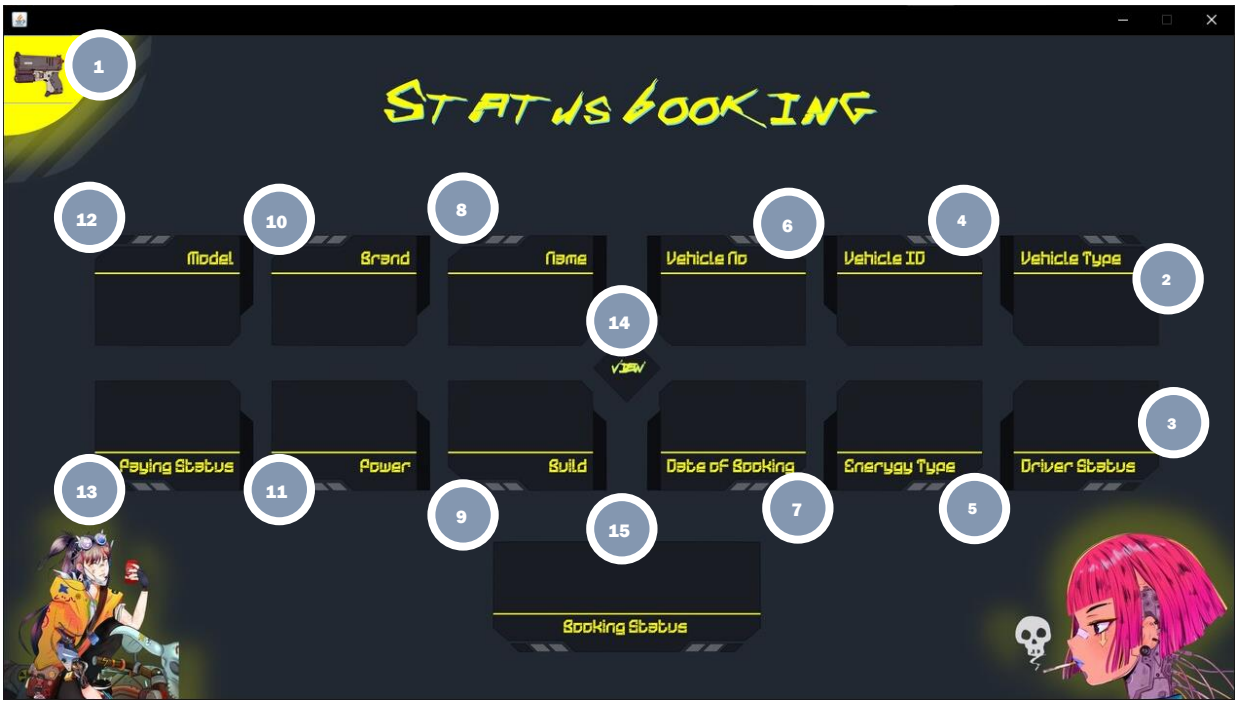


The customer can access this page from the customer list. Before he can click "Change Password," the customer will need to enter a new password. The customer can change the username in the same way with the user name. Click on "Change username" after entering a new username.

Num	Name	Palette	Function
1	jButton5	Button	Return to the previous page
2	jTextField3	TextField	where the customer can write his/her new password
3	jTextField2	TextField	where the customer can write his/her new username
4	jButton1	Label	the customer click here to confirm changing the password
5	jButton2	Label	the customer click here to confirm changing the username


CHANGE PASSWORD & USERNAME
change password
change username
NEW Password
123
NEW Username
Abdule

Step 7



Finally, the customer can access the "Status Booking" or partial customer section via the customer menu. The page will initially be devoid of data, but when he clicks the view button, the data will begin to appear as shown below.

Num	Name	Palette	Function
1	jButton5	Button	Return to the previous page
2	jLabel1	Label	where the vehicle type of the customer will be displayed
3	jLabel2	Label	where the Driver of the customer will be displayed
4	jLabel3	Label	where the vehicle ID of the customer will be displayed
5	jLabel4	Label	where the Enerygy Type of the customer will be displayed
6	jLabel5	Label	where the Vehicle No of the customer will be displayed
7	jLabel6	Label	where the Date of Booking of the customer will be displayed
8	jLabel7	Label	where the Name of the customer will be displayed
9	jLabel8	Label	where the Build of the customer will be displayed
10	jLabel9	Label	where the Brand of the customer will be displayed

11	jLabel10	Label	where the Power of the customer will be displayed
12	jLabel11	Label	where the Model of the customer will be displayed
13	jLabel12	Label	where the Paying Status of the customer will be displayed
14	jLabel13	Label	where the Booking Status of the customer will be displayed
15	jButton1	Button	you can click here to display all customer's status booking

❖ JFRAME & CODE

❖ ADD ACCOUNT

❖

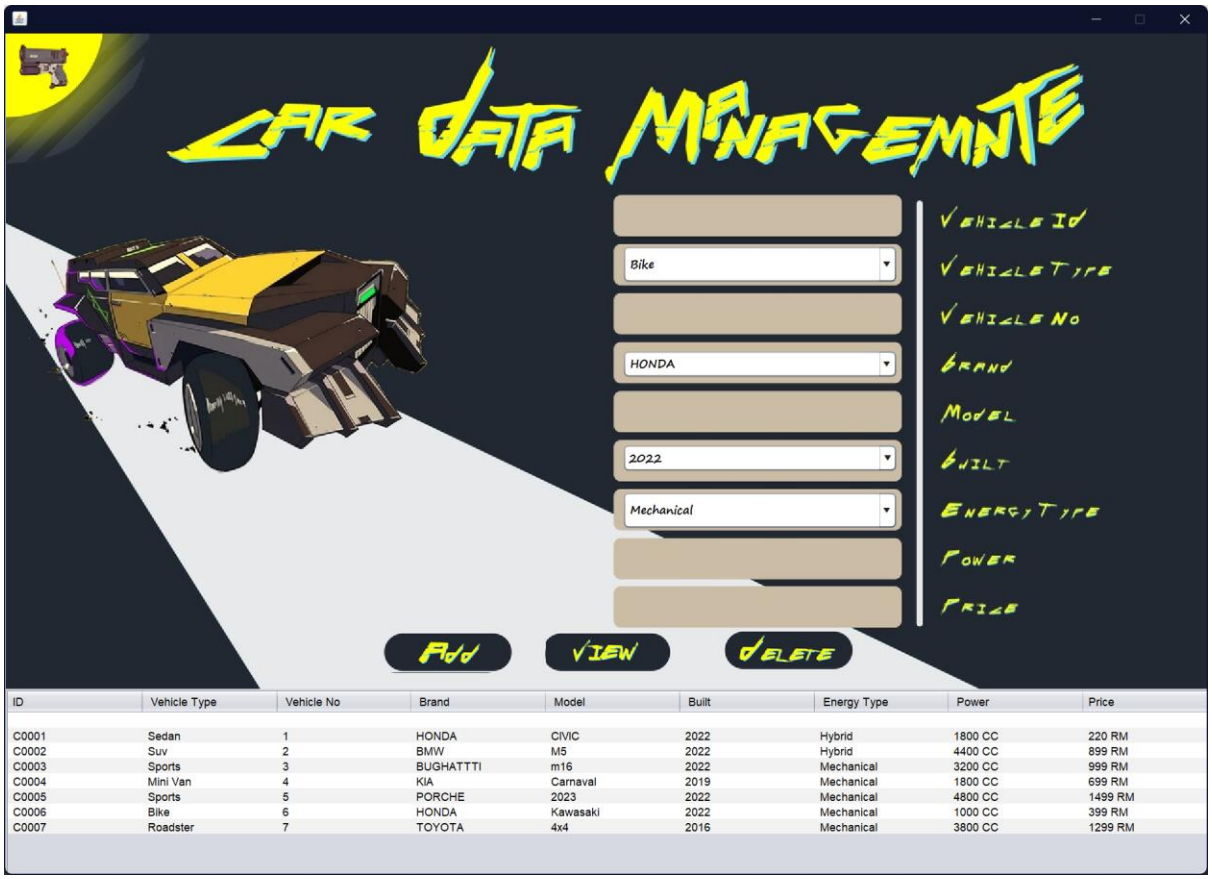
```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String full_name=jTextField2.getText().toString();
    String gender = null;
    String Acc_type = null;
    if(jRadioButton1.isSelected()){
        gender=  "male";
    }
    else if (jRadioButton2.isSelected())
    {
        gender = "female";

    }

    else {
        JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid Data Input", title: "Alert", messageType: JOptionPane.WARNING_MESSAGE);
    }

    if(jRadioButton3.isSelected()){
        Acc_type = "AD";
    }
    else if(jRadioButton4.isSelected())
    {
        Acc_type = "CU";

    }

    else {
        JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid Data Input", title: "Alert", messageType: JOptionPane.WARNING_MESSAGE);
    }
```

❖

```
    String age=jComboBox1.getSelectedItem().toString();
    String national_id=jTextField3.getText().toString();
    String email_address=jTextField4.getText().toString();
    String password=jPasswordField2.getText().toString();
    String Confirm_password=jPasswordField3.getText().toString();

    if       (password.equals( anObject: Confirm_password)){
        try{
            FileWriter Writer=new FileWriter( fileName: "Customer_Data.txt", append: true);
            Writer write(full_name+"/"+age+"/"+gender+"/"+national_id+"/"+email_address+"/"+password+"/"+Acc_type+"\n");
            //Writer.write(System.getProperty("line.separator"));
            Writer.close();
            JOpticnPane.showMessageDialog( parentComponent: null, message: "Data Added Successfully");
            setVisible( b: false);
            new User_Login().setVisible( b: true);

        }
        catch(Exception e){
        }
    }
    else {
        JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid Data Input", title: "Alert", messageType: JOptionPane.WARNING_MESSAGE);

    }
}

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Admin_Options obj=new Admin_Options();
    obj.setVisible( b: true);
    dispose();
}
```

❖

❖ ADD CARS



ID	Vehicle Type	Vehicle No	Brand	Model	Built	Energy Type	Power	Price
C0001	Sedan	1	HONDA	CIVIC	2022	Hybrid	1800 CC	220 RM
C0002	Suv	2	BMW	M5	2022	Hybrid	4400 CC	899 RM
C0003	Sports	3	BUGHATTTI	m16	2022	Mechanical	3200 CC	999 RM
C0004	Mini Van	4	KIA	Carnaval	2019	Mechanical	1800 CC	699 RM
C0005	Sports	5	PORCHE	2023	2022	Mechanical	4800 CC	1499 RM
C0006	Bike	6	HONDA	Kawasaki	2022	Mechanical	1000 CC	399 RM
C0007	Roadster	7	TOYOTA	4x4	2016	Mechanical	3800 CC	1299 RM

❖

❖

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
```

```
public class Add_Cars extends javax.swing.JFrame {

    public void addData(){
    JFrame f=new JFrame();
    String id=jTextField3.getText().toString();
    String vehicleType=jComboBox1.getSelectedItem().toString();
    String vehicleNumber=jTextField1.getText().toString();
    String brand=jComboBox4.getSelectedItem().toString();
    String model=jTextField4.getText().toString();
    String built=jComboBox2.getSelectedItem().toString();
    String energyType=jComboBox3.getSelectedItem().toString();
    String power=jTextField2.getText().toString();
    String price = jTextField5.getText().toString();
    try{
    FileWriter Writer=new FileWriter( fileName: "Car_Data.txt", append: true);
    Writer.write(id+"/"+vehicleType+"/"+vehicleNumber+"/"+brand+"/"+model+"/"+built+"/"+energyType+" /"+power+" CC"+ " /"+price+" RM" +"\n");
   //Writer.write(System.getProperty("line.separator"));
    Writer.close();
    JOptionPane.showMessageDialog( parentComponent: null, message: "Data Added Successfully");
    setVisible( b: false);
    new Add_Cars().setVisible( b: true);

    }
    catch(Exception e){
    }


    }
```

❖

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    DefaultTableModel model = (DefaultTableModel) jTable1.getModel();

    // get selected row index

    try{
        int SelectedRowIndex = jTable1.getSelectedRow();
        model.removeRow( row: SelectedRowIndex);
    }catch(Exception ex)
    {
        JOptionPane.showMessageDialog( parentComponent: null,  message: ex);
    }
}
```

❖

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String filePath="Car_Data.txt";
    File file=new File( pathname: filePath);
    try{
        BufferedReader br=new BufferedReader(new FileReader(file));
        String firstLine=br.readLine().trim();
        String[] columnsName=firstLine.split( regex: ",");
        DefaultTableModel model=(DefaultTableModel)jTable1.getModel();
        model.setColumnIdentifiers( newIdentifiers: columnsName);

        String line = "";
        while(line != null) {
            line = br.readLine().trim();
            if(line != null) {
                Object[] dataRow = line.split( regex: "/");
                model.addRow( rowData: dataRow);
            }
        }

    }
    catch(Exception ex){
        Logger.getLogger( name: Add_Cars.class.getName()).log( level: Level.SEVERE, msg: null, thrown: ex);
    }

}
```

❖

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    addData();
```

❖

```
private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Admin_Options obj=new Admin_Options();
    obj.setVisible( b: true);
    dispose();
```

❖

❖ ADMIN OPTIONS

❖

```
import javaapplication1.Add_Cars;
import javaapplication1.Welcome;
import javaapplication1.Manage_Bookings;
```

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Manage_Bookings obj=new Manage_Bookings();
    obj.setVisible( b: true);
    dispose();
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Report obj=new Report();
    obj.setVisible( b: true);
    dispose();


}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Add_Cars obj=new Add_Cars();
    obj.setVisible( b: true);
    dispose();
}

private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Add_Account obj=new Add_Account();
    obj.setVisible( b: true);
    dispose();
}

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Welcome obj=new Welcome();
    obj.setVisible( b: true);
    dispose();
}
```

❖

❖ BOOK APPOINTMENT

❖ 

❖

❖

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
```

❖

❖

```
public class Book_Appointment extends javax.swing.JFrame {
String data,name;
```

❖

```
public Book_Appointment() {

    initComponents();

}

public Book_Appointment(String Uname) {
    this.name = Uname;

    initComponents();

}
```

❖

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

String date=jDateChooser2.getCalendar().getTime().toString();
String driver=jComboBox1.getSelectedItem().toString();
TableModel model1 = jTable1.getModel();
int indexs[] = jTable1.getSelectedRows();
Object row[] = new Object[8];
 for(int i = 0; i < indexs.length; i++)
   {
       row[0] = model1.getValueAt(indexs[i], columnIndex: 0);
       row[1] = model1.getValueAt(indexs[i], columnIndex: 1);
       row[2] = model1.getValueAt(indexs[i], columnIndex: 2);
       row[3] = model1.getValueAt(indexs[i], columnIndex: 3);
       row[4] = model1.getValueAt(indexs[i], columnIndex: 4);
       row[5] = model1.getValueAt(indexs[i], columnIndex: 5);
       row[6] = model1.getValueAt(indexs[i], columnIndex: 6);
       row[7] = model1.getValueAt(indexs[i], columnIndex: 7);
   }


 String ID, Vehicle_Type, Vehicle_no, Brand,Model, built, Energy_Type,Power;
 ID = row[0].toString();
 Vehicle_Type = row[1].toString();
 Vehicle_no = row[2].toString();
 Brand = row[3].toString();
 Model = row[4].toString();
 built = row[5].toString();
 Energy_Type = row[6].toString();
 Power = row[7].toString();


 data=(this.name+","+ID+","+Vehicle_Type+","+Vehicle_no+","+Brand+","+Model+","+built+","+Energy_Type+","+Power+","+date+","+driver+","+ "PAID"+","+ "Waiting").toString();

  try{
  FileWriter Writer=new FileWriter( fileName: "Booking_Data.txt", append: true);
  Writer.write(data+"\n");
 //Writer.write(System.getProperty("line.separator"));
  Writer.close();
  JOptionPane.showMessageDialog( parentComponent: null, message: "Booking Confirmed & Paid");



  }
  catch(Exception e){
  }
```

❖

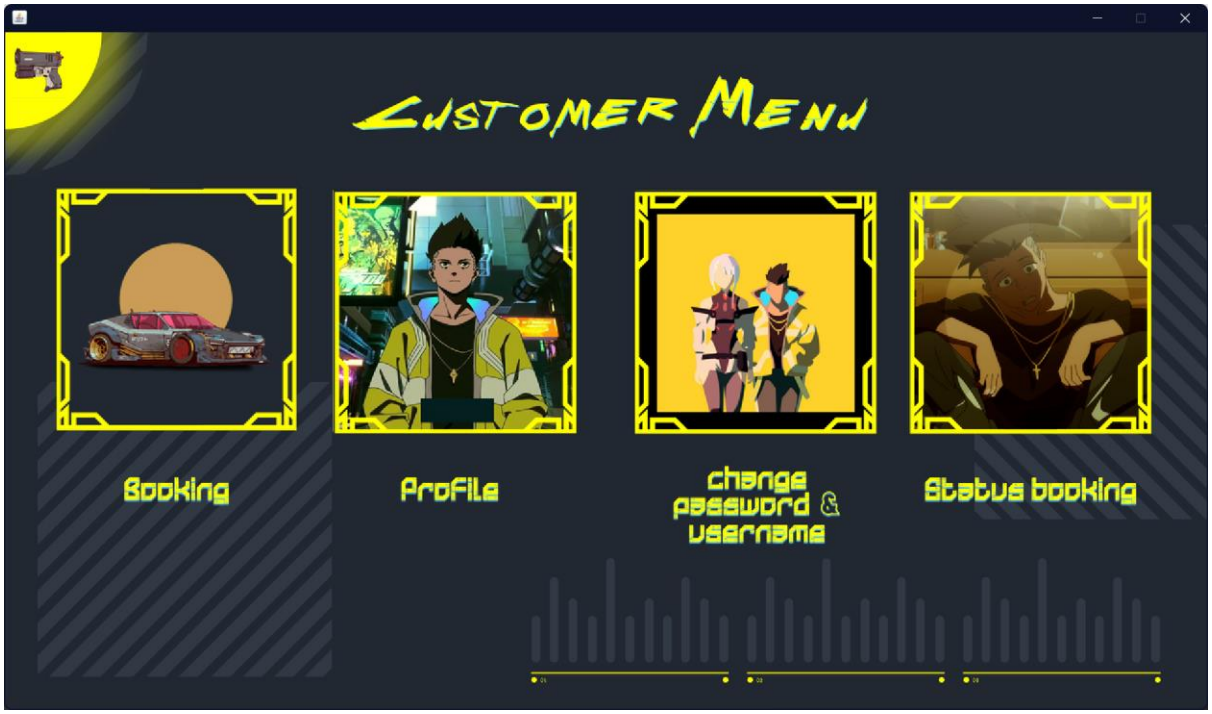
```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    CU_Options obj=new CU_Options( Uname: name);
    obj.setVisible( b: true);
    dispose();
}
```

❖

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String filePath="Car_Data.txt";
    File file=new File( pathname: filePath);
    try{
        BufferedReader br=new BufferedReader(new FileReader(file));
        String firstLine=br.readLine().trim();
        String[] columnsName=firstLine.split( regex: ",");
        DefaultTableModel model=(DefaultTableModel)jTable1.getModel();
        model.setColumnIdentifiers( newIdentifiers: columnsName);

        String line = "";
        while(line != null) {
            line = br.readLine().trim();
            if(line != null) {
                Object[] dataRow = line.split( regex: "/");
                model.addRow( rowData: dataRow);
            }
        }

    }
    catch(Exception ex){
        Logger.getLogger( name: Add_Cars.class.getName()).log( level: Level.SEVERE, msg: null, thrown: ex);
    }

}
```

❖

❖ CU OPTIONS.

❖ 

❖
```
 */
public class CU_Options extends javax.swing.JFrame {
    String name;
```

❖
```
public CU_Options(String Uname){
    this.name = Uname;
    initComponents();
}
```

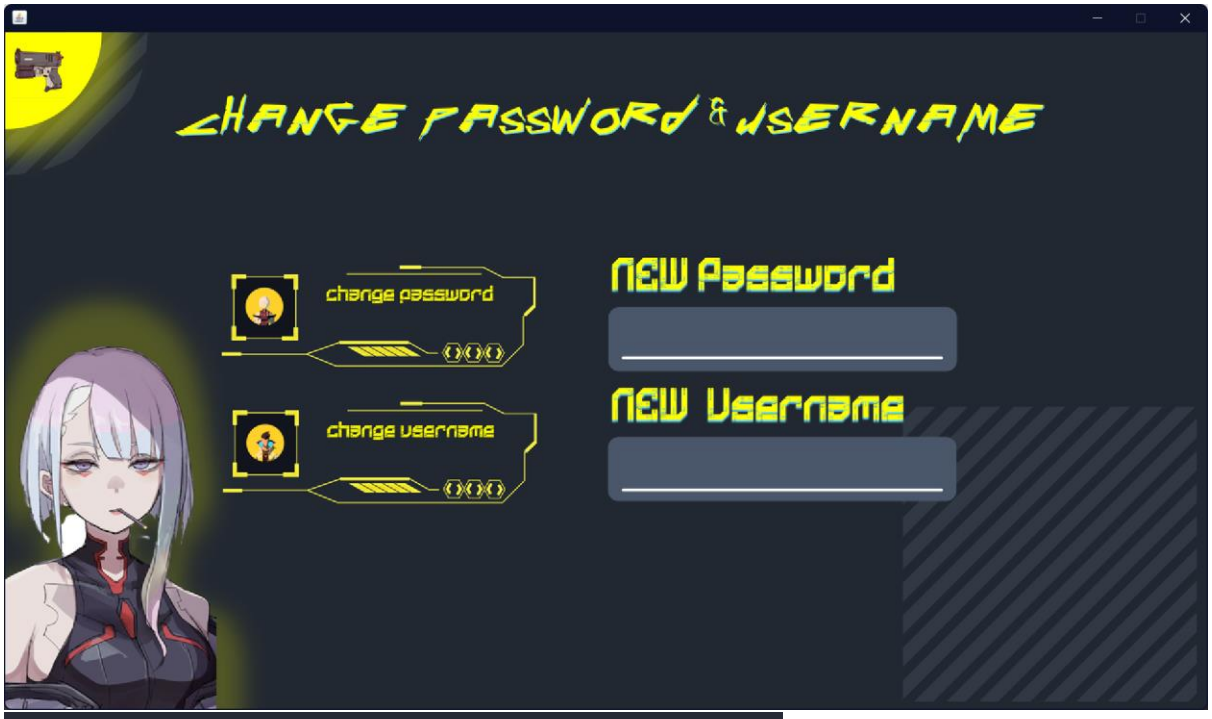
```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    dispose();
    Book_Appointment obj=new Book_Appointment( Uname: name);
    obj.setVisible( b: true);
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    dispose();
    Profile obj=new Profile( Uname: name);
    obj.setVisible( b: true);
}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    dispose();
    Change_Pass_User obj=new Change_Pass_User( Uname: name);
    obj.setVisible( b: true);
}

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    dispose();
    Status_booking obj=new Status_booking( Uname: name);
    obj.setVisible( b: true);
}

private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Welcome obj=new Welcome();
    obj.setVisible( b: true);
    dispose();
}
```

❖

❖ CAR RENTAL SYSTEM.

```
package javaapplication1;

/**
 *
 * @author Wajahat Qazi
 */
public class Car_Rental_System {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
      Welcome abc=new Welcome();
       abc.setVisible( b: true);
    }

}
```

❖ CHANGE PASSWORD USERNAME

❖ 

❖
```
import java.io.BufferedWriter;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import javax.swing.JOptionPane;
import java.io.FileWriter;
```

❖

```java
/**
 * Creates new form Change_Pass_User
 */
public Change_Pass_User() {
    initComponents();
}
public Change_Pass_User(String Uname) {
    this.name = Uname;
    initComponents();
}
```

❖

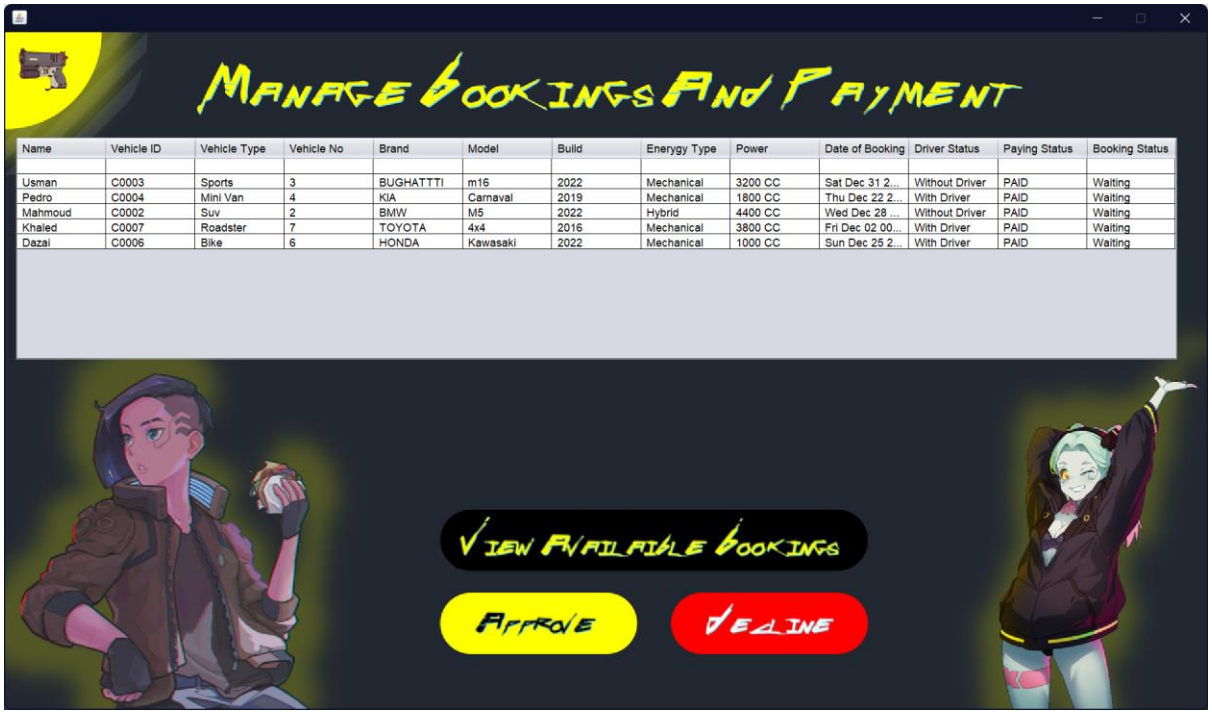
```java
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    CU_Options obj=new CU_Options( Uname: name);
    obj.setVisible( b: true);
    dispose();
}
```

❖

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    File file  = new File( pathname: "Customer_Data.txt");
    try {
        BufferedReader br = new BufferedReader (new FileReader (file) ) ;
        String [] rec;
        String Status_old, Status_New;
        Object [] allRec = br . lines () . toArray () ;
        for (int i =1; i < allRec. length; i++) {
            rec = allRec [i] . toString () . trim() .split( regex: "/");
            if (rec[0].equals( anObject: name)) {
                Status_old = rec[5];
                Status_New = jTextField3.getText();
                rec[5] = Status_New;
                String newrec="";
                for (int j=0; j<rec. length; j++) {
                    newrec = newrec+rec [j]+"/";
                }
                allRec [i]= newrec;

            }

        }
        BufferedWriter bw = new BufferedWriter (new FileWriter (file) ) ;
        for (int i=0; i<allRec. length; i++) {
            System.out.println("Rec["+i+"]: "+allRec[i]) ;
            bw . write (allRec[i]+"\n") ;


        }
        bw . close ( ) ;
        br . close () ;
       JOptionPane.showMessageDialog( parentComponent: null, message: "Password Had Been Reset Successfully");




    } catch (Exception e) {
    }

}
```

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    File file  = new File( pathname: "Customer_Data.txt");
    try {
        BufferedReader br = new BufferedReader (new FileReader (file) ) ;
        String [] rec;
        String Status_old, Status_New;
        Object [] allRec = br . lines () . toArray () ;
        for (int i =1; i < allRec. length; i++) {
            rec = allRec [i] . toString () . trim() .split( regex: "/");
            if (rec[0].equals( anObject: name)) {
                Status_old = rec[0];
                Status_New = jTextField2.getText();
                rec[0] = Status_New;
                String newrec="";
                for (int j=0; j<rec. length; j++) {
                    newrec = newrec+rec [j]+"/";
                }
                allRec [i]= newrec;

            }

        }

         BufferedWriter bw = new BufferedWriter (new FileWriter (file) ) ;
        for (int i=0; i<allRec. length; i++) {
            System.out.println("Rec["+i+"]: "+allRec[i]) ;
            bw . write (allRec[i]+"\n") ;


        }
        bw . close ( ) ;
        br . close () ;
       JOptionPane.showMessageDialog( parentComponent: null, message: "User name Had Been Reset  Successfully");



    } catch (Exception e) {
    }
}
```

❖ MANAGE BOOKINGS.

❖ 

❖

```
import java.io.BufferedWriter;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.*;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;

/**
```

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:

        String filePath="Booking_Data.txt";
         File file=new File( pathname: filePath);


         try{
             BufferedReader br=new BufferedReader(new FileReader(file));
             String firstLine=br.readLine().trim();
             String[] columnsName=firstLine.split( regex: "/");
             DefaultTableModel model=(DefaultTableModel)jTable1.getModel();
             model.setColumnIdentifiers( newIdentifiers: columnsName);


             String line = "";
while(line != null) {
    line = br.readLine().trim();
    if(line != null) {
         Object[] dataRow = line.split( regex: ",");
         model.addRow( rowData: dataRow);
    }
}

         }
         catch(Exception ex){

         }


    }
```

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
      DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
     TableModel model1 = jTable1.getModel();
     int SelectedRowIndex = jTable1.getSelectedRow();
 int indexs[] = jTable1.getSelectedRows();
 Object row[] = new Object[12];
  for(int i = 0; i < indexs.length; i++)
    {
      row[0] = model1.getValueAt(indexs[i],  columnIndex: 0);
      row[1] = model1.getValueAt(indexs[i],  columnIndex: 1);
      row[2] = model1.getValueAt(indexs[i],  columnIndex: 2);
      row[3] = model1.getValueAt(indexs[i],  columnIndex: 3);
      row[4] = model1.getValueAt(indexs[i],  columnIndex: 4);
      row[5] = model1.getValueAt(indexs[i],  columnIndex: 5);
      row[6] = model1.getValueAt(indexs[i],  columnIndex: 6);
      row[7] = model1.getValueAt(indexs[i],  columnIndex: 7);
      row[8] = model1.getValueAt(indexs[i],  columnIndex: 8);
      row[9] = model1.getValueAt(indexs[i],  columnIndex: 9);
      row[10] = model1.getValueAt(indexs[i],  columnIndex: 10);
      row[11] = model1.getValueAt(indexs[i],  columnIndex: 11);
    }
  String name =(String) row[0];

  String arr=Arrays.toString( a: row);

File file  = new File( pathname: "Booking_Data.txt");
    try {
        BufferedReader br = new BufferedReader (new FileReader (file) ) ;
        String [] rec;
        String Status_old, Status_New;
        Object [] allRec = br . lines () . toArray () ;
        for (int i =1; i < allRec. length; i++) {
            rec = allRec [i] . toString () . trim() .split( regex: ",");
            if (rec[0].equals( anObject: name)) {
                Status_old = rec[12];
                Status_New = "Declined & Refunded";
                rec[12] = Status_New;
                String newrec="";
                for (int j=0; j<rec. length; j++) {
                    newrec = newrec+rec [j]+",";
                }
                allRec [i]= newrec;

            }

        }
        BufferedWriter bw = new BufferedWriter (new FileWriter (file) ) ;
        for (int i=0; i<allRec. length; i++) {
            System.out.println("Rec["+i+"]: "+allRec[i]) ;
            bw . write (allRec[i]+"\n") ;


        }
        bw . close ( ) ;
        br . close () ;
       JOptionPane.showMessageDialog( parentComponent: null, message: "Booking Declined & Refunded Successfully");



    } catch (Exception e) {
    }
```

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

    TableModel model1 = jTable1.getModel();
 int indexs[] = jTable1.getSelectedRows();
 Object row[] = new Object[12];
  for(int i = 0; i < indexs.length; i++)
    {
        row[0] = model1.getValueAt(indexs[i],  columnIndex: 0);
        row[1] = model1.getValueAt(indexs[i],  columnIndex: 1);
        row[2] = model1.getValueAt(indexs[i],  columnIndex: 2);
        row[3] = model1.getValueAt(indexs[i],  columnIndex: 3);
        row[4] = model1.getValueAt(indexs[i],  columnIndex: 4);
        row[5] = model1.getValueAt(indexs[i],  columnIndex: 5);
        row[6] = model1.getValueAt(indexs[i],  columnIndex: 6);
        row[7] = model1.getValueAt(indexs[i],  columnIndex: 7);
        row[8] = model1.getValueAt(indexs[i],  columnIndex: 8);
        row[9] = model1.getValueAt(indexs[i],  columnIndex: 9);
        row[10]= model1.getValueAt(indexs[i],  columnIndex: 10);
        row[11]= model1.getValueAt(indexs[i],  columnIndex: 11);
    }

  String aa=Arrays.toString( a: row);
  String name =(String) row[0];

  File file  = new File( pathname: "Booking_Data.txt");
   try {
       BufferedReader br = new BufferedReader (new FileReader (file) ) ;
       String [] rec;
       String Status_old, Status_New;
       Object [] allRec = br . lines () . toArray () ;
       for (int i =1; i < allRec. length; i++) {
          rec = allRec [i] . toString () . trim() .split( regex: ",");
          if (rec[0].equals( anObject: name)) {
              Status_old = rec[12];
              Status_New = "Approved";
              rec[12] = Status_New;
              String newrec="";
              for (int j=0; j<rec. length; j++) {
                 newrec = newrec+rec [j]+",";
              }
              allRec [i]= newrec;

          }

       }
       BufferedWriter bw = new BufferedWriter (new FileWriter (file) ) ;
       for (int i=0; i<allRec. length; i++) {
          System.out.println("Rec["+i+"]: "+allRec[i]) ;
          bw . write (allRec[i]+"\n") ;


       }
       bw . close ( ) ;
       br . close () ;
      JOptionPane.showMessageDialog( parentComponent: null, message: "Booking Approved Successfully");



   } catch (Exception e) {
   }
```

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Admin_Options obj=new Admin_Options();
    obj.setVisible( b: true);
    dispose();

}
```

❖

❖ PROFILE

❖ 

❖
```
import java.io.File;
import java.util.Scanner;
```

❖
```
public class Profile extends javax.swing.JFrame {
    String name;
```

```
    /**
     * Creates new form Profile
     */
    public Profile() {
        initComponents();
    }
    public Profile(String Uname) {
        this.name = Uname;
        initComponents();
    }
```

❖

```
    private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
       CU_Options obj=new CU_Options( Uname: name);
        obj.setVisible( b: true);
        dispose();
    }
```

❖

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String password,email,na,age,gender;
    password = "";
    na = "";
    age = "";
    gender = "";
    email = "";

    try {
        Scanner in = new Scanner(new File( pathname: "Customer_Data.txt"));
        while (in.hasNextLine()){
            String s = in.nextLine();
            String[] sArray = s.split( regex: "/");
            if (this.name.equals(sArray[0]) ){
                age = sArray[1].toString();
                gender = sArray[2].toString();
                na = sArray[3].toString();
                email = sArray[4].toString();
                password = sArray[5].toString();
                 jLabel4.setText( text: password);
                 jLabel5.setText( text: this.name);
                 jLabel2.setText( text: na);
                 jLabel6.setText( text: email);
                 jLabel7.setText( text: age);
                 jLabel3.setText( text: gender);
                break;

            }

        }


    } catch (Exception e) {
    }
}
```

❖

❖ REPORT

❖ 

❖
```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import javax.swing.table.DefaultTableModel;
```

❖
```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Admin_Options obj=new Admin_Options();
    obj.setVisible( b: true);
    dispose();
}
```

❖

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
         String filePath="Booking_Data.txt";
        File file=new File( pathname: filePath);


        try{
            BufferedReader br=new BufferedReader(new FileReader(file));
            String firstLine=br.readLine().trim();
            String[] columnsName=firstLine.split( regex: "/");
            DefaultTableModel model=(DefaultTableModel)jTable1.getModel();
            model.setColumnIdentifiers( newIdentifiers: columnsName);


            String line = "";
while(line != null) {
    line = br.readLine().trim();
    if(line != null) {
         Object[] dataRow = line.split( regex: ",");
         model.addRow( rowData: dataRow);
    }
}

        }
        catch(Exception ex){

        }

    }
```

❖

❖ SECRET

❖ 

❖
```
import java . io . File;
import javax . sound . sampled . AudioInputStream;
import javax . sound . sampled . AudioSystem;
import javax . sound . sampled.Clip;
```

❖

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
        try{

          File musicPath = new File ( pathname: "y2mate_com_Cyberpunk_Edgerunners_Soundtrack_I_Really_Want_to_Stay.wav") ;

          if (musicPath. exists () )
          {
                    AudioInputStream audioInput = AudioSystem.getAudioInputStream ( file: musicPath) ;
                    Clip clip = AudioSystem. getClip () ;
                    clip.open ( stream: audioInput) ;
                    clip.start() ;

          }
          else
          {
                    System. out . println ( x: "can't find file") ;
          }
}
catch (Exception e)
{

}
  }
```

❖

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Welcome obj=new Welcome();
    obj.setVisible( b: true);
    dispose();
}
```

❖

❖ STATUS BOOKING

❖ 

❖
```
import java.io.File;
import java.util.Scanner;
```

❖
```
public class Status_booking extends javax.swing.JFrame {
    String name;
```

```
/**
 * Creates new form Status_booking
 */
public Status_booking() {
    initComponents();
}
public Status_booking(String Uname) {
    this.name = Uname;
    initComponents();
}
```

❖

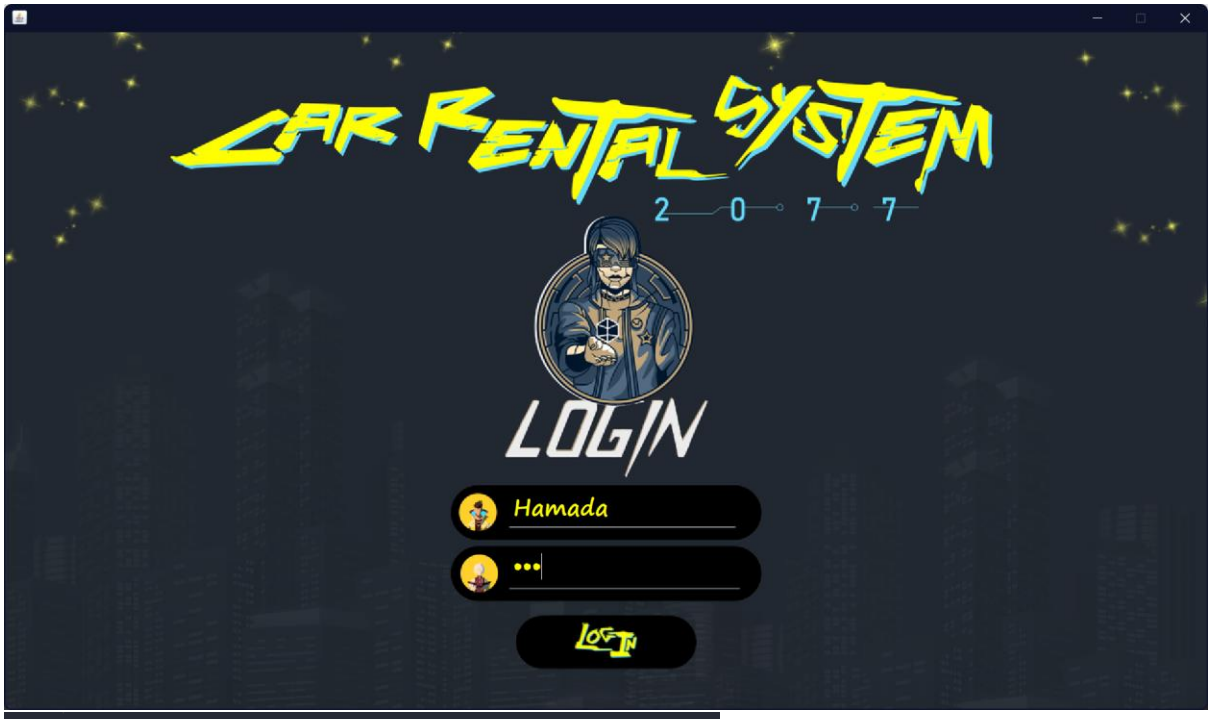
```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    CU_Options obj=new CU_Options( Uname: name);
    obj.setVisible( b: true);
    dispose();
}
```

❖

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String Vehicle_ID , Vehicle_Type , Vehicle_No, Brand, Model, Build,Enerygy_Type, Power, Date_of_Booking, Driver_Status, Paying_Status, Booking_Status;
    Vehicle_No = "";
    Brand = "";
    Build = "";
    Booking_Status = "";
    Model = "";
    Vehicle_Type = "";
    Vehicle_ID = "";
    Enerygy_Type = "";
    Power= "";
    Date_of_Booking= "";
    Driver_Status= "";
    Paying_Status= "";

    try {
        Scanner in = new Scanner(new File( pathname: "Booking_Data.txt"));
        while (in.hasNextLine()){
            String s = in.nextLine();
            String[] sArray = s.split( regex: ",");
            if (this.name.equals(sArray[0]) ){
                Vehicle_ID = sArray[1].toString();
                Vehicle_Type = sArray[2].toString();
                Vehicle_No = sArray[3].toString();
                Brand = sArray[4].toString();
                Model = sArray[5].toString();
                Build = sArray[6].toString();
                Enerygy_Type = sArray[7].toString();
                Power = sArray[8].toString();
                Date_of_Booking = sArray[9].toString();
                Driver_Status = sArray[10].toString();
                Paying_Status = sArray[11].toString();
                Booking_Status = sArray[12].toString();


                jLabel2.setText( text: Vehicle_Type);
                jLabel3.setText( text: Vehicle_No);
                jLabel4.setText( text: this.name);
                jLabel5.setText( text: Brand);
                jLabel6.setText( text: Model);
                jLabel7.setText( text: Paying_Status);
                jLabel8.setText( text: Power);
                jLabel9.setText( text: Build);
                jLabel10.setText( text: Date_of_Booking);
                jLabel11.setText( text: Enerygy_Type);
                jLabel12.setText( text: Driver_Status);
                jLabel13.setText( text: Booking_Status);
                jLabel14.setText( text: Vehicle_ID);


                break;

            }

        }
    } catch (Exception e) {
    }
}
```

❖

❖ USER LOGIN

❖ 

❖
```
package javaapplication1;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
import javax.swing.JOptionPane;
```

❖
```
public class User_Login extends javax.swing.JFrame {
String name;
```

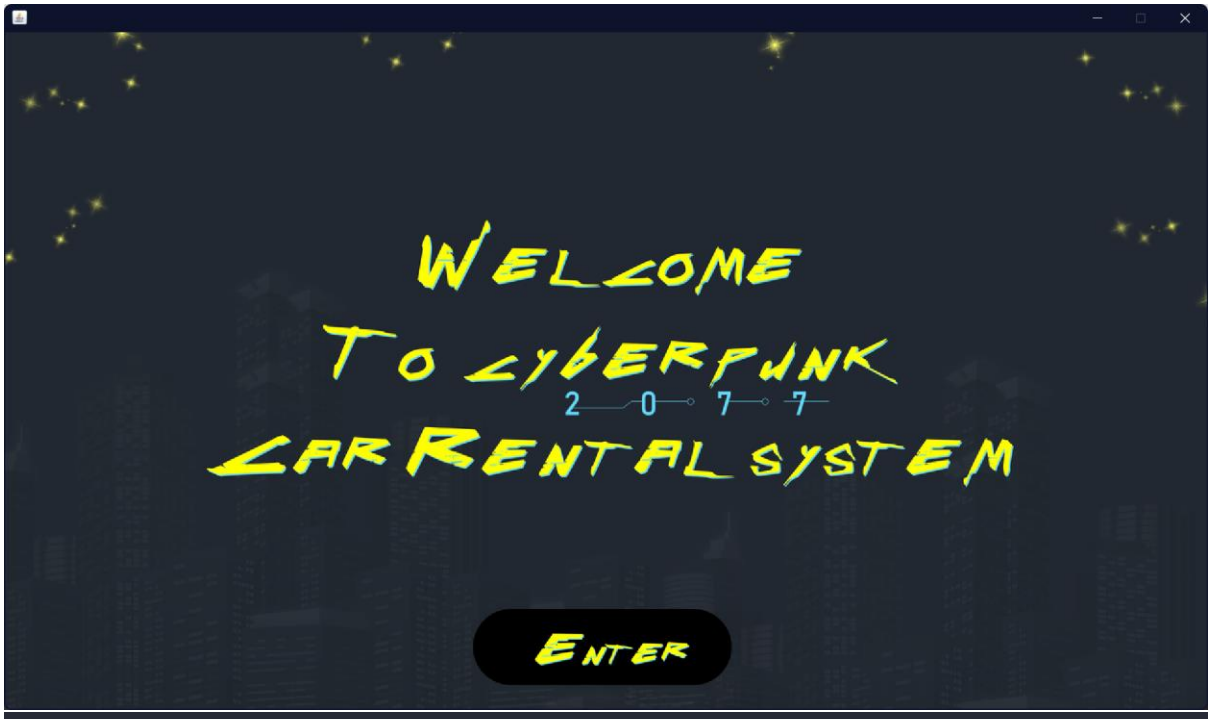
❖
```
 public User_Login() {

     initComponents();
 }
```

❖

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    name=jTextField1.getText();
    String password=jPasswordField1.getText().toString();


    try {
        Scanner in = new Scanner(new File( pathname: "Customer_Data.txt"));
        while (in.hasNextLine()){
            String s = in.nextLine();
            String[] sArray = s.split( regex: "/");

            if (name.equals(sArray[0]) && password.equals(sArray[5]) && sArray[6].equals( anObject: "CU")){
                JOptionPane.showMessageDialog( parentComponent: null,  message: "Login Successful",  title: "Success", messageType: JOptionPane.INFORMATION_MESSAGE);
                dispose();
                CU_Options obj=new CU_Options( Uname: name);

                obj.setVisible( b: true);
               break;
            }
            else if (name.equals(sArray[0]) && password.equals(sArray[5]) && sArray[6].equals( anObject: "AD")){
                JOptionPane.showMessageDialog( parentComponent: null,  message: "Login Successful",  title: "Success", messageType: JOptionPane.INFORMATION_MESSAGE);
                dispose();
                Admin_Options obj=new Admin_Options();
                obj.setVisible( b: true);
                break;
            }
             else if (name.equals( anObject: "CYBERPUNK") && password.equals( anObject: "CYBERPUNK") ){
                JOptionPane.showMessageDialog( parentComponent: null,  message: "Login Successful",  title: "Success", messageType: JOptionPane.INFORMATION_MESSAGE);
                dispose();
                Secret obj=new Secret();
                obj.setVisible( b: true);
               break;
             }

        }

        in.close();


    }
    catch (FileNotFoundException e) {
        JOptionPane.showMessageDialog( parentComponent: null,
                message: "User Database Not Found",  title: "Error",
                messageType: JOptionPane.ERROR_MESSAGE);
    }


}
```

❖

❖ WELCOME

❖ 

❖

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    User_Login obj=new User_Login();
    obj.setVisible( b: true);
    dispose();
}
```

❖

❖ CONCLUSION

- The system was constructed so that it would fulfil the requirements mentioned earlier successfully. Due to the limitation of not using a database, which would have made things more professional and accessible, we created and developed this system to fulfil those functional and non-functional needs, as mentioned earlier. In doing so, we were able to meet all of the requirements. More features can also be added, but since we are running short of time, we cannot accommodate them.

❖ NOTE IF THERE IS ANY IMPORTING ERROR

Its Probably because of jCalender in Book appointment page u need to install this plugin

https://www.youtube.com/watch?v=robcQaF-jfM

this a video will help u

❖ REFERENCES

- MyCodeSpace. (2020, November 14). Java full project(Inventory management system) [Video]. YouTube. https://www.youtube.com/watch?v=HfSQL2H7_mE&t=6173s

- Knowledge to Share. (2019, March 3). JTable in JAVA Swing | Update selected row from JTable [Video]. YouTube. https://www.youtube.com/watch?v=Tg62AxNRir4

- Ahmad Hanis. (2018, November 18). Java GUI with JTable and Arraylist [Video]. YouTube. https://www.youtube.com/watch?v=UQt3piMATSw

- GeeksforGeeks. 2020. Classes And Objects In Java - Geeksforgeeks. [online] Available at:
- Classes and Objects in Java - GeeksforGeeks [Accessed 26 November 2020].