CAR RENTAL SERVICE

[1]Kumari Puja, [2]Gaurav Srivastava, [3]Yashwanth Pokala

[1]Information Systems, Northeastern University
[2]Information Systems, Northeastern University
[3]Information Systems, Northeastern University

{puja.k, srivastava.g, pokala.y}@northeastern.edu

Abstract:

The application we have developed is an efficient solution that addresses the needs of the rental services industry. By utilizing Eclipse IDE, Java, JavaFX, and Scene Builder, we have created an application that allows rental services to list their available vehicles and enables customers to manage their bookings with ease.

The model classes for booking and car serve as the backbone of the application, managing the data that is entered into the system. The Controller classes, on the other hand, play a crucial role in managing the interaction between the UI and backend, ensuring that data is displayed accurately, and that user input is processed correctly.

To create an intuitive and visually appealing user interface, we have utilized Scene Builder to design the UI. The result is a user-friendly application that is easy to navigate and simplifies the rental process for customers.

Our application comprises 8 controller classes and 3 model classes, indicating a well-structured and organized codebase.

Overall, the application we have developed is a reliable and efficient solution that provides rental services and customers with an easy-to-use interface for managing vehicle rentals.

Keywords—Business Layer(Model Classes), Middle Layer(Controller Classes), User Interface, Inheritance, Interfaces, HashMap, List and Collections.

I. Problem Description

The rental services industry has long been dominated by private companies that provide rental services through their own applications, limiting access to the market for those without the resources to develop their own application. This exclusion creates a barrier for those with vehicles willing to rent them out but lack the means to do so.

To address this issue, we have developed an application that allows anyone with a vehicle willing to rent it to list their car on our platform. Our application serves as a solution to the lack of access to the rental services market by providing a platform that is accessible to a wider audience.

The challenge of creating a platform that is open to all, regardless of resources, is significant. Our application provides a user-friendly interface that enables individuals to manage their vehicle rentals and simplifies the rental process for customers. By utilizing Model and Controller classes and designing a visually appealing UI using Scene Builder, our application provides a well-structured and organized codebase that ensures accurate data management and user input processing.

Overall, the problem we aimed to address was the exclusion of individuals from the rental services market due to limited resources. Our application serves as a solution to this problem by providing a platform that is accessible to anyone with a vehicle willing to rent it out, simplifying the rental process for all parties involved.

II. Analysis (Related Work)

ZipCar's dominant market position has led to high pricing for consumers in the car rental industry, with a lack of competition and high costs associated with maintaining a large fleet of vehicles being the primary reasons. Additionally, the complicated GUI of existing car rental company apps has made the user experience difficult for many consumers. Our application aims to address these issues by providing an alternative pricing structure where the cost of renting a car is decided by the car owner and a simplified GUI for a better user experience.

Furthermore, the high pricing and complicated GUI of existing car rental company apps may lead to a sense of distrust among consumers, as they may feel that they are not getting the best value for their money or that the rental company is not transparent in its pricing. By providing a more transparent pricing structure and a simplified GUI, our application seeks to build trust with consumers and establish a loyal customer base. This will not only benefit our application but also the car owners who are providing their vehicles for rent, as they can attract more customers through a trustworthy platform.

III. System Design

The System Design consists of three Layers, Business layer(Model Classes), Middle Layer(Controller Classes) and User Interface.

1. **User Interface Design:** The user interface (UI) is designed using Scene Builder, which is a visual layout

tool. The UI is designed to be user-friendly and visually appealing to attract users. It includes options for renting a vehicle, searching for available vehicles, and adding vehicles.

The user interface consists of 6 fxml files.

a. login-view.fxml: The login page
b. adminUser-addNewBrand.fxml: Admin page for adding vehicle brands
c. adminUser-add.fxml: Admin page for adding vehicles from the existing brands.
d. adminUser-view.fxml: Admin page for viewing vehicles by searching the number plate in the search box.
e. customer-new-booking.fxml: Customer page for booking vehicles.
f. customer-search-booking.fxml: Customer can search existing bookings.

2. **Model Classes:** The application includes three model classes – BookingModel.java, CarModel.java and TableBookingModel.java - that manage the data entered into the system.

   The application includes three model classes –

   a. The **BookingModel** class stores the details of the booking made by the user.
   b. **CarModel** class stores the details of the vehicle listed by the owner.
   c. The **TableBookingModel** class stores the list of all the vehicles booked by the customer in an array list and displays it in a table

3. **Controller Classes:** Controller classes manage the interaction between the UI and the backend. The application includes eight controller classes that handle user input, process data, and update the UI.

   These controller classes include BookingController, CarController, UserController, SearchController, PaymentController, ConfirmationController, DashboardController, and NavigationController

4. **Concepts used:**

   a. JavaFX
   b. Class Definition
   c. Inheritance
   d. Interface
   e. Lists
   f. Maps
   g. Collections

IV. Implementation

A. *Business Layer(Model Classes):*

The rental services application consists of three main model classes – CarModel, BookingModel and TableBooking.

1.1 CarModel:

The CarModel class represents a vehicle that is available for rent. The CarModel consists of four objects, they are plateNumber, carMake, carYear and carColor. The getters and setters methods are defined along with the explicit constructors.

```
public CarModel(String plateNumber, String carMake, String carYear, String carColor) {
this.plateNumber = plateNumber;
this.carMake = carMake;
this.carYear = carYear;
this.carColor = carColor;
}
```

The above piece of code is the constructor defined for CarModel class.

1.3 BookingModel:

The BookingModel class has three objects: String bookingId, LocalDate dateFrom and LocalDate dateTo. ***LocalDate*** *is an immutable date-time object that represents a date, often viewed as year-month-day.*

The BookingModel class inherits the CarModel class. As an instance of a Booking also includes the vehicle booked.

```
public BookingModel(CarModel carModel, String bookingId, LocalDate dateFrom,LocalDate dateTo){
this.setPlateNumber(carModel.getPlateNumber());
this.setCarMake(carModel.getCarMake());
this.setCarYear(carModel.getCarYear());
this.setCarColor(carModel.getCarColor());
this.bookingId=bookingId;
this.dateFrom=dateFrom;
this.dateTo=dateTo;
}
```

The above piece of code is the constructor defined for BookingModel class.

1.3 TableBookingModel:

The BookingModel class has three objects: carMake, carYear, carColor and plateNumber. The data type of all the objects are SimpleStringProperty.

This class is used to store the booking details in an ArrayList and display it in a table.

```
public TableBookingModel(String carMake, String carYear, String carColor,String plateNumber)
{
this.carMake = new SimpleStringProperty(carMake);
this.carYear = new SimpleStringProperty(carYear);
this.carColor = new
SimpleStringProperty(carColor);
this.plateNumber=new
SimpleStringProperty(plateNumber);
}
```

The above piece of code is the constructor defined for TableBookingModel class.

B. ***Middle-Layer:***

2.1 LoginController

The LoginController class defines the behavior of the login screen. When the user clicks on the "Login" button, the user's input is verified by calling the verifyUserDetails() method. If the username and password fields are empty, an error message is displayed. If the input is valid and matches one of the pre-defined usernames and passwords, the loggedInUser variable is set to the entered username, and the user is redirected to the appropriate screen based on their user type. If the input is invalid, an error message is displayed. This class is implemented using the JavaFX framework and is part of the CarRentalServiceApp.

2.2 ***AdminController:***

This code defines an ***AdminController*** *class that is responsible for handling the different button clicks in the administrator user interface of a car rental service application.*
The ***AdminController*** *has methods for handling button clicks, including* ***viewCarButtonClick()***, ***addCarButtonClick()***, ***addNewBrandButtonClick()***, *and* ***logOutButtonClick()***.
The ***viewCarButtonClick()*** *method updates the screen to show the view of all the available cars,* ***addCarButtonClick()*** *updates the screen to show the form for adding a new car,* ***addNewBrandButtonClick()*** *updates the screen to show the form for adding a new brand, and* ***logOutButtonClick()*** *logs the user out of the application and updates the screen to show the login view.*
The ***logOutButtonClick()*** *method displays a confirmation alert dialog to confirm that the user wants to log out. If the user clicks the OK button, then the currently logged-in user is set to an empty string, and the screen is updated to show the login view.*

2.3 ***AdminAddBrandController***

The AdminAddBrandController class is responsible for handling the addition of a new car brand by the administrator. The class includes FXML fields for the brand name and code, as well as a warning label to alert the user in case of errors.
The resetBrandButtonClick() method is triggered when the user clicks on the "Reset" button, which clears the brand name and code fields and hides the warning label.
The saveBrandButtonAction() method is called when the user clicks on the "Save" button to add a new car brand. It first checks whether both brand name and code fields are filled or not. If one of them is empty, it sets the warning label to "Please Enter Brand Name" or "Please Enter Brand Code" accordingly. If both fields are filled, it checks whether the brand code already exists in the carMakeMap, which is a HashMap containing all the car brands added so far. If the brand code already exists, the warning label is set to "Brand Already Exists", and if not, the new brand is added to the carMakeMap using the brand code as the key and brand name as the value, and the warning label is set to "Brand Saved!!!".

2.4 ***AdminAddCarController***

The ***AdminAddCarController*** *class is responsible for handling user input for adding a new car to the system. It has fields for the car's make, year, color, and license plate number, and a label to display any warnings or error messages.*
When the user clicks the "Save Car" button, the ***saveCarButtonAction()*** *method is called. This method first checks that all the necessary fields have been filled out, and if not, displays an appropriate error message. Otherwise, it checks if a car with the same license plate number already exists in the system. If so, it displays an error message. If not, it creates a new* ***CarModel*** *object with the provided information, adds it to the* ***cars*** *map and* ***carList***, *and displays a success message.*
The ***resetButtonClick()*** *method is called when the user clicks the "Reset" button. It simply clears all fields and hides any warning/error messages.*
The ***initialize()*** *method sets up the* ***carMakeCombo*** *and* ***carYearCombo*** *dropdown menus with the available options. The available car makes are obtained from the* ***carMakeMap***, *and sorted alphabetically before being added to the* ***carMakeCombo***. *The available years are obtained from the* ***carYearList*** *and added to the* ***carYearCombo***.

2.5 ***AdminUserViewController***

The ***AdminUserViewController*** *is a JavaFX controller class that extends the* ***AdminController*** *class and implements the* ***Initializable*** *interface. The purpose of this class is to control the functionality of the* ***adminUser-view.fxml*** *file.*

*The class contains several **@FXML**-annotated fields that correspond to different labels and text fields defined in the FXML file. There is also a **searchButtonAction()** method that searches for a car in the **cars** map based on the plate number entered in the **searchTextBox** field.*
*If a car is found, the **toggleResultPanel()** method is called to display the car's details in the appropriate labels. If no car is found, a warning message is displayed.*
*In the **initialize()** method, if the **cars** map is empty, a sample car is added to it and its details are displayed in the corresponding labels. The **toggleResultPanel()** method is also called to hide the car details until a search is performed.*

*2.6 **CustomerController:***

*The code defines a **CustomerController** class which is responsible for handling events for the customer user interface. It contains three methods: **searchBookingButtonClick()**, **addNewBookingButtonClick()**, and **logOutButtonClick()**.*
*The **searchBookingButtonClick()** method opens the "customer-search-booking.fxml" view, which allows the customer to search for an existing booking. The **addNewBookingButtonClick()** method opens the "customer-new-booking.fxml" view, which allows the customer to create a new booking.*
*The **logOutButtonClick()** method logs out the user by displaying a confirmation dialog box with a message asking if they are sure they want to exit. If the user confirms, the application sets the **loggedInUser** variable to an empty string and returns the user to the login view.*

*2.7 **CustomerNewBookingController:***

The CustomerNewBookingController class is responsible for handling the logic of the new booking process for a customer in a car rental service application. It has the following functionality:

- *Display a form where the customer can search for available cars by selecting the make, year, and dates.*
- *Display a list of available cars in a table, which can be sorted by make, year, and color.*
- *Allow the customer to select a car from the list and create a new booking for that car for the selected dates.*
- *Display a message with the booking ID for the customer to reference.*
- *Handle any validation errors and display appropriate error messages to the user.*

The implementation makes use of JavaFX components, such as ComboBox, TableView, and DatePicker, to build the user interface. It also makes use of data models such as CarModel, BookingModel, and TableBookingModel to represent the data used in the application. The data is stored in various lists and maps, such as carList, bookingList, and bookingMap. The implementation uses various checks and filters to ensure that the selected car is available for the selected dates, and creates a new booking object when the user clicks the book button.

*2.8 **CustomerSearchBookingController:***

The code is a Java class named "CustomerSearchBookingController" which implements the "Initializable" interface. It contains a number of instance variables, including labels, text fields, and a map of bookings. The "searchButtonAction" method is called when the user clicks on a search button. If the search text field is empty, a warning message is displayed. Otherwise, if the booking map contains the entered booking ID, the details of the corresponding booking are displayed. If not, a "No Data Found" message is displayed. The "toggleResultPanel" method is used to show or hide the booking details panel depending on whether a valid booking has been found. The "initialize" method is called when the controller is initialized and sets the username label, hides the search warning label, and hides the booking details panel. Overall, the class provides a graphical user interface for searching and displaying car rental booking details.

C. *User-Interface:*

The application consists of six FXML files, each representing a different view in the application.

- *"adminUser-add.fxml" represents the view for adding a new car to the system.*
- *"adminUser-addNewBrand.fxml" represents the view for adding a new brand of car to the system.*
- *"adminUser-view.fxml" represents the view for viewing the details of all cars in the system.*
- *"customer-new-booking.fxml" represents the view for a customer to create a new booking.*
- *"customer-search-booking.fxml" represents the view for a customer to search for their existing bookings.*
- *"login-view.fxml" represents the view for the login page of the application.*

These FXML files are used to build the graphical user interface of the application and define the layout and components of each view.

V. Evaluation

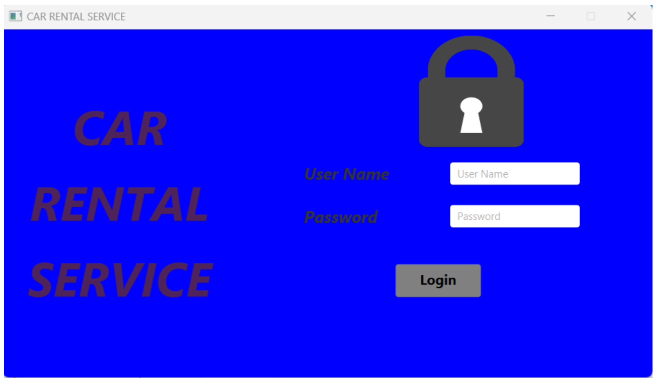


Fig 1.1 Login Page

Fig 1.1 shows the login page of the application, the login page is common for both admin and customer. By default the login credentials for Admin is username:admin and password:admin.

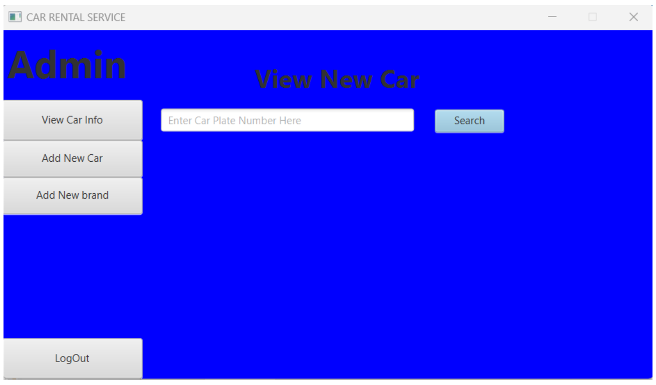
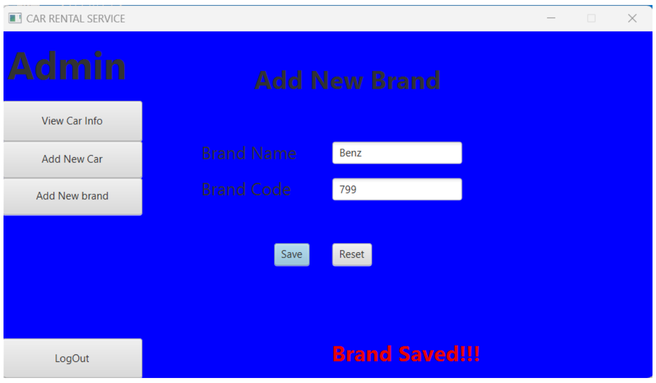


Fig 1.2 Admin Home Page

The Fig 1.2 shows the home page of Admin. The admin can view car information, add new car brands and add new car from the added brands.



Brand Name Benz
Brand Code 799
Brand Saved!!!

Fig 1.3 Admin Page for adding new brands

As you can see in the above figure, the admin can add brand by
Specifying the Brand name and brand code and clicking the save button. In the above case we have added a new brand named benz with code 799.

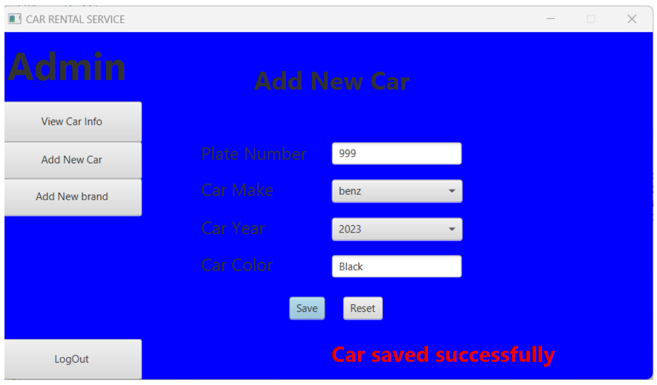
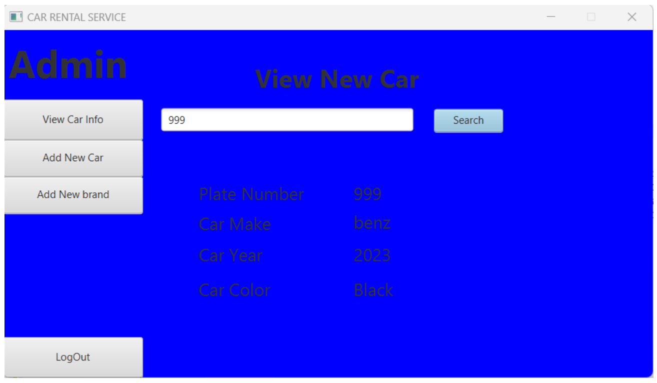


Plate Number 999
Car Make benz
Car Year 2023
Car Color Black
Car saved successfully

Fig 1.4 Admin Page for adding new cars

This page helps the admin to add new cars. The car make, that is the car brand can be selected from the already added brands. The Car Make is a drop down menu.

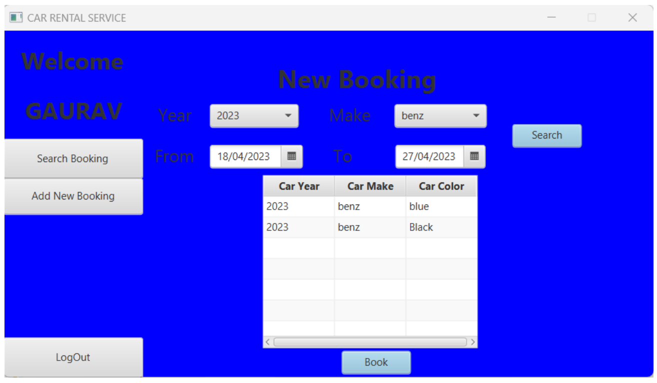


999

Plate Number 999
Car Make benz
Car Year 2023
Car Color Black

Fig 1.5 Admin Page for viewing cars

In Fig 1.4 you can see that, by searching the car number(number plate) in the search bar we can have the car details. In this case, the previously entered car is searched.

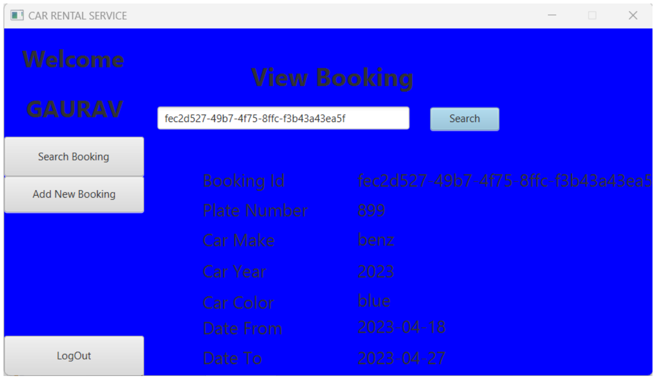


Year 2023 Make benz
From 18/04/2023 To 27/04/2023

Car Year	Car Make	Car Color
2023	benz	blue
2023	benz	Black

Fig 1.6 Customer Page for booking cars

As you can see in the figure, the customer can search for the cars available for booking based on the filters applied and book them. Once a car is booked a bookingId is generated.



fec2d527-49b7-4f75-8ffc-f3b43a43ea5f

Booking Id	fec2d527-49b7-4f75-8ffc-f3b43a43ea5
Plate Number	899
Car Make	benz
Car Year	2023
Car Color	blue
Date From	2023-04-18
Date To	2023-04-27

Fig 1.7 Customer Page for viewing the cars booked

As shown in Fig 1.7 the customer was able to get the booking details by searching the bookingId in the search bar.

VII. Discussion (Reflection)

The car rental application implemented in Java has proved to be a useful tool for both the admin and the customers. The admin is able to easily manage the different brands and their respective cars in the system. The addition of new cars and brands can be easily done through the use of forms provided by the system. The admin can also easily search for a particular car in the system using the car number. This feature helps the admin to quickly locate a car and make necessary changes such as updating its availability status.

On the customer side, the application provides a user-friendly interface for searching and booking cars. Customers can easily apply filters to narrow down their search based on their specific requirements such as the make and year of the car. The system displays a table with a list of available cars that meet the selected criteria. Customers can then select a car and book it by entering the required information. Upon successful booking, the system generates a unique booking ID which can be used by the customer to track the status of their booking.

One of the notable advantages of this application is its ability to provide real-time updates on car availability. This helps to minimize the chances of double bookings and ensures that customers have accurate information on the availability of cars. Additionally, the application provides a secure login system for both admin and customers, ensuring that only authorized users have access to the system.

VIII. Conclusions and Future Work

In conclusion, the Java-based car rental application has proven to be a reliable and efficient tool for managing car rental services. The application provides a user-friendly interface for both the admin and customers, allowing for easy navigation and management of the system. The application's ability to provide real-time updates on car availability helps to minimize errors and ensure a smooth booking process for customers. The secure login system adds an extra layer of security, ensuring that only authorized personnel have access to the system.

The advantage of this type of application is anyone can list their vehicles for rentals, and there is no need for a third-party to manage the vehicles.

The Project for now is limited to only one customer/user. If given more time we would have expanded the scope of the project. By allowing different customers to register and allowing the customer to both list their car for rent or rent an already listed car. This would make our application more useful.

IX. Job Assignment

- Member 1(Yashwanth Pokala): Model Classes (Business Layer), User Interface, Report.
- Member 2(Gaurav Srivastava): Controller Classes, Main class.
- Member 3(Kumari Puja): User Interface, PPT, Controller Classes.

References

[1] Oracle JavaFX documentation https://docs.oracle.com/javase/8/javafx/api/toc.htm