

Software design

Documentation

Assignment 2

Name: Cristian Muntean

Group:30434

Contents

1. Objective	3
1.1. General requirement.....	3
1.2. Specific requirement	3
2. Tools used.....	4
2.1. List of used tools	4
2.2. Justification for using the selected tools.....	4
2.2.1. Java.....	4
2.2.2. Java Swing	4
2.2.3. Sergeds library.....	4
2.2.4. Gradle	5
2.2.5. JUnit, Groovy and Spock	5
3. UML diagrams	5
3.1. Use case diagram	5
3.2. Class diagram	7
3.3. Entity-Relationship diagram	9
4. Implementation	10
4.1. Login window	10
4.2. Employee window	10
4.3. Manager window	13
4.4. Admin window	14

1. Objective

The objective of this topic is to familiarize with the Model-View-Presenter architectural pattern. For information persistence, a relational database (SQL Server, MySQL, etc.) will be used.

1.1. General requirement

The general requirement for the first homework is:

- ❖ During the analysis phase, a use case diagram will be created.
- ❖ During the design phase, the class diagram will be created following the MVVM architecture (which uses the Command behavioral design pattern) and the GRASP principles, as well as the entity-relationship diagram corresponding to the database.
- ❖ During the implementation phase, code will be written to implement all the functionalities specified by the use case diagram using:
 - the design provided by the class diagram
 - one of the following programming languages: C#, C++, Java, Python.
- ❖ During the testing phase, unit tests (testing project) corresponding to the operations of creating the database, establishing a connection to the database, creating tables, and querying tables of the database will be implemented.
- ❖ Completion of the topic will consist of submitting a directory containing:
 - A file with the UML diagrams created;
 - The database;
 - The software application;
 - Documentation (minimum of 10 pages) - a file that includes:
 - the student's name, group;
 - the statement of the problem;
 - the tools used;
 - the justification for the chosen programming language;
 - description of the UML diagrams;
 - description of the application.

1.2. Specific requirement

The requirement for the specific homework is:

Develop an application that can be used in an auto service. The client application will have 3 types of users: employee, manager, and administrator.

After authentication, employee users can perform the following operations:

- ❖ View a list of all existing vehicles in the service, sorted by brand and fuel type;
- ❖ Filter vehicles based on certain criteria: owner, brand, color, fuel type;
- ❖ Search for a vehicle by owner;
- ❖ CRUD operations regarding the persistence of vehicles and their owners.
- ❖ Save lists with information about vehicles in multiple formats: csv, json, xml, txt.

After authentication, manager users can perform the following operations:

- ❖ View a list of all existing vehicles in the service, sorted by brand and fuel type;
- ❖ Filter vehicles based on certain criteria: owner, brand, color, fuel type.
- ❖ Search for a vehicle by owner
- ❖ Save lists with information about vehicles in multiple formats: csv, json, xml, txt.

After authentication, administrator users can perform the following operations:

- ❖ CRUD operations for user-related information within the application;
- ❖ View the list of all users and filter it by user type.

2. Tools used

2.1. List of used tools

For the implementation of the application I used several tools:

- Java – for the development of the production code which is run by the application
- Java Swing – for the graphical user interface
- Sergeds swing mvvm – library for creating the mvvm pattern
- Gradle – for dependency management
- JUnit, Groovy and Spock – for writing the unit tests

2.2. Justification for using the selected tools

2.2.1. Java

I decided to use Java for the development of the production code in order to try to understand better how different architectural patterns and principles are applied in said programming language. During the development of this project, I learned how to implement the MVP architectural pattern and how to apply the SOLID principles in order to create classes and method which are more comprehensible and easier to understand.

2.2.2. Java Swing

I used Java Swing for the graphical user interface because I was already familiar with it and had some experience in creating different layouts of GUIs in order to present the required data in a user friendly way.

2.2.3. Sergeds library

I used this external library in order to create the MVVM architectural pattern in swing. This library offers several bindings that can be used to bind the text fields, combo boxes and check boxes from the view classes in the view-model classes.

2.2.4. Gradle

I recently learned Gradle from my workplace, so I decided to try to deepen my knowledge in it and try to make a clear separation between Maven and Gradle in order to create an opinion on which one I would prefer to use day by day.

2.2.5. JUnit, Groovy and Spock

Knowing that the implementation of the application was going to be created in Java, I used JUnit to create the unit tests for the production code.

I also recently learned Groovy and Spock at my workplace. I discovered that writing tests in Groovy makes the tests more readable and are easier to understand what is a “prerequisite” for a test, what is actually tested and what is expected from a test. Also, Groovy provides more readable test names that

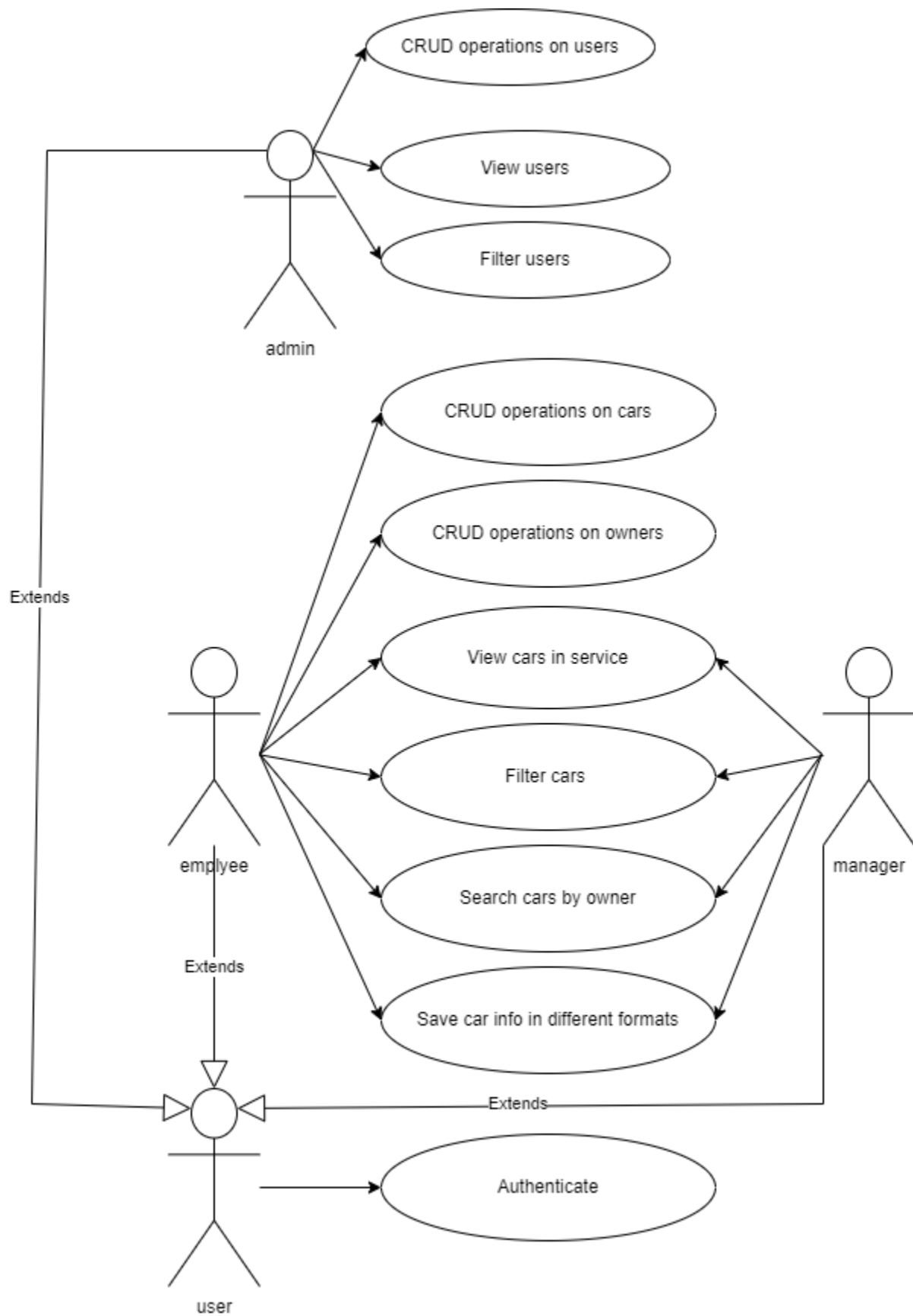
On top of that, Spock helped to remove redundant code, which would have been copy-pasted in multiple tests which tested similar functionalities. For example, when testing if a specific table is created after running a given method, the expected result should be that they are created. This test should be done for every table in the database, which would have the same structure: call method which creates a table, then check if it exists. By using the “where” from Spock, this redundancy is removed and a single test is created which will be iterated over using different parameters for functions.

3. UML diagrams

During the analysis and the design phase, the use case diagram, the class diagram and the entity-relationship diagram had to be created.

3.1. Use case diagram

The first one was the use case diagram, which was created during the analysis phase of the project.



We can see that in order to use the application, the user must first authenticate into it. After the authentication is done, based on their role, the user will be able to perform some given actions.

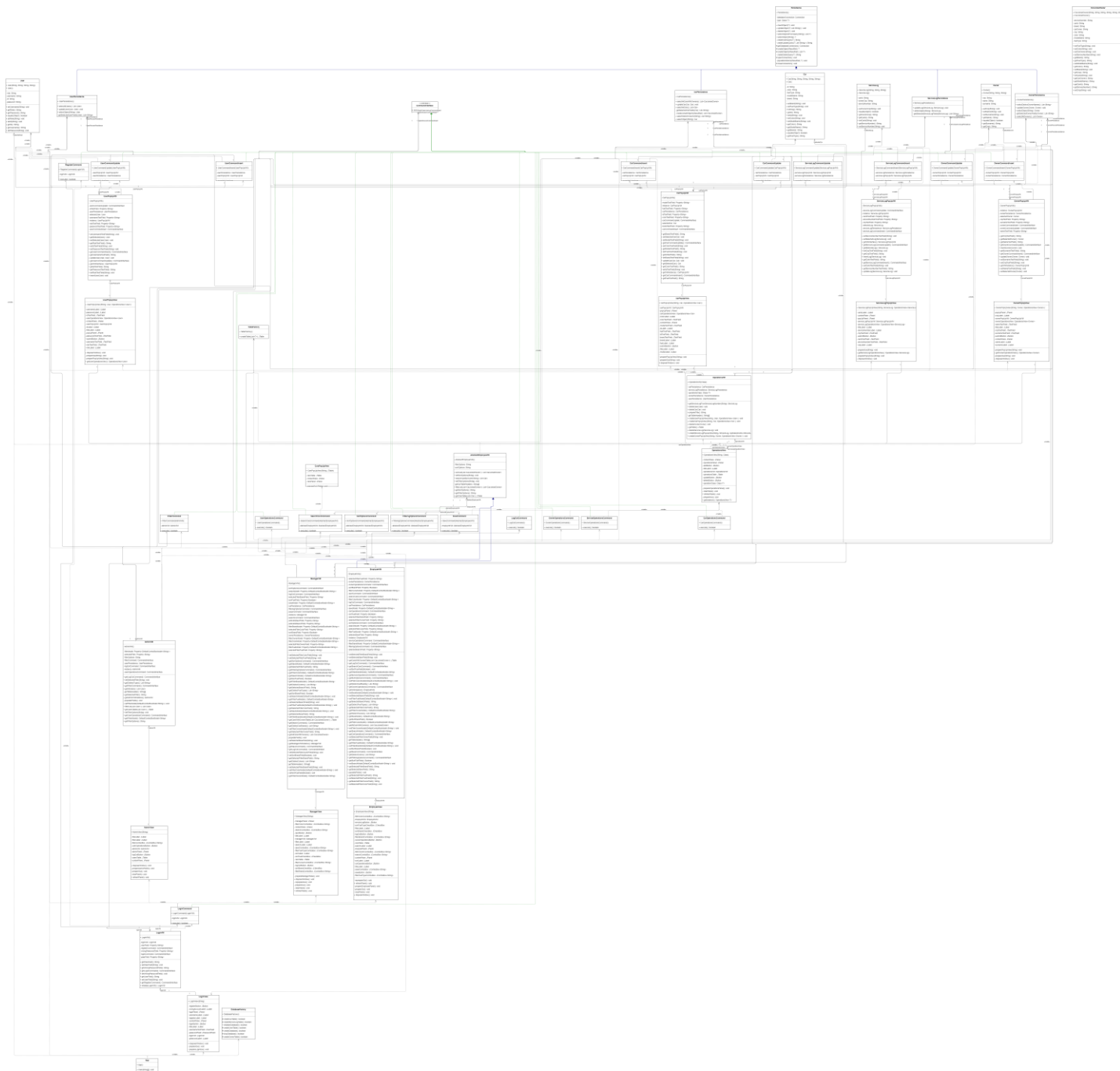
If the user is an employee, he will be able to perform Create, Read, Update and Delete operations on cars and owners of the cars, in order to create reservation in the service shop. Also, he can see the cars that are currently in the shop, sort them by brand and fuel type and filter them by owner, car brand, fuel type and color of the car. Also, the employee is able to search cars by the owner of the car and save the car information in different formats (csv, xml, json, txt).

If the user is a manager, he will only be able to see the cars that are available in the shop, sort them by brand and fuel type and filter them by owner, car brand, fuel type and color of the car. Also, the employee is able to search cars by the owner of the car and save the car information in different formats (csv, xml, json, txt) just like the employee can.

If the user is an admin, he will be able to view the users that are present in the database, filter them by the role, perform Create, Update and Delete operations on the current users of the application and see the users that are currently registered into the application's database.

3.2. Class diagram

During the design phase of the project, the class diagram was created which shows the classes of the application and the relationship between them.



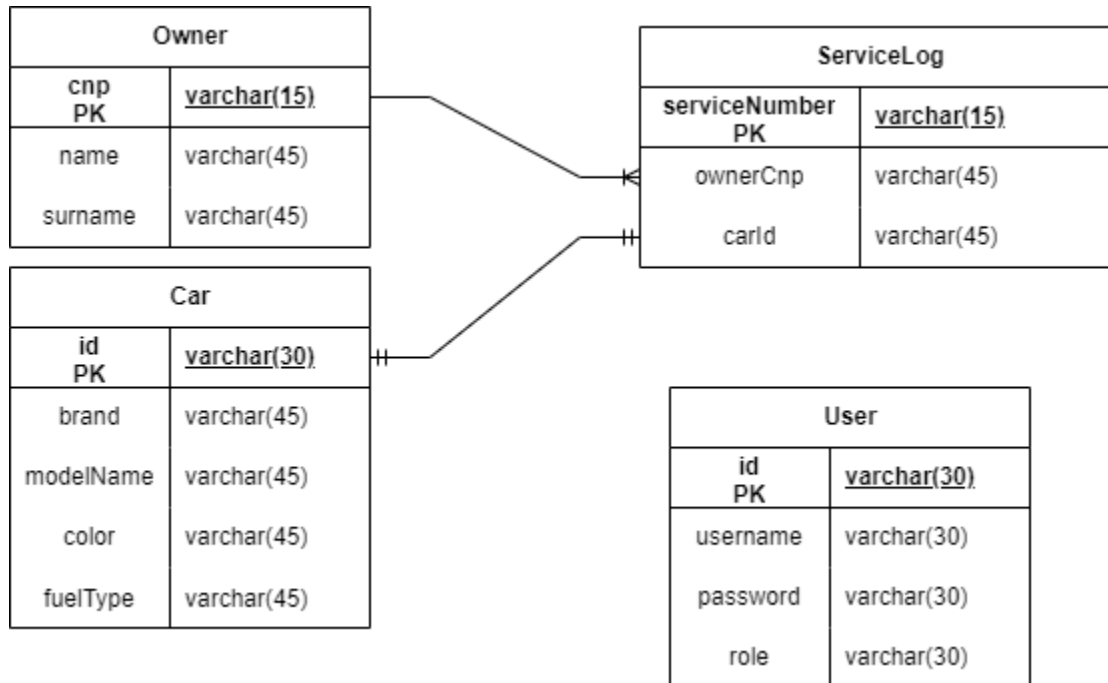
In this class diagram we can see that the MVVM architectural pattern was respected, each class being part of one of the following packages: model, persistence, view or view-model. In order for the user to perform an action at a given point in time, each element from the view classes (that has an action associated to it) has an action listener that, when triggered, a command from its associated view-model is executed. The View-Model classes contain properties that are bound to elements from the view classes and commands that should be executed whenever an action is triggered.

Furthermore, every time the application starts, it will try to create a new database if doesn't currently exist. On every operation that requires the access to the database, first a connection will be established,

then the operation will be performed, then the connection will be closed. This is done in order to promote the efficiency of the application.

3.3. Entity-Relationship diagram

The entity-relationship diagram was also created during the design phase of the project. It represents the structure of the database and the relationships that are present between different tables.



The User table will store information about all the users. Each user has a unique id and username, a password and a role. The role can be one of the following 3 options: "employee", "manager" or "admin".

The Owner table will store information about each owner of a car. Every owner has a unique cnp, a name and a surname.

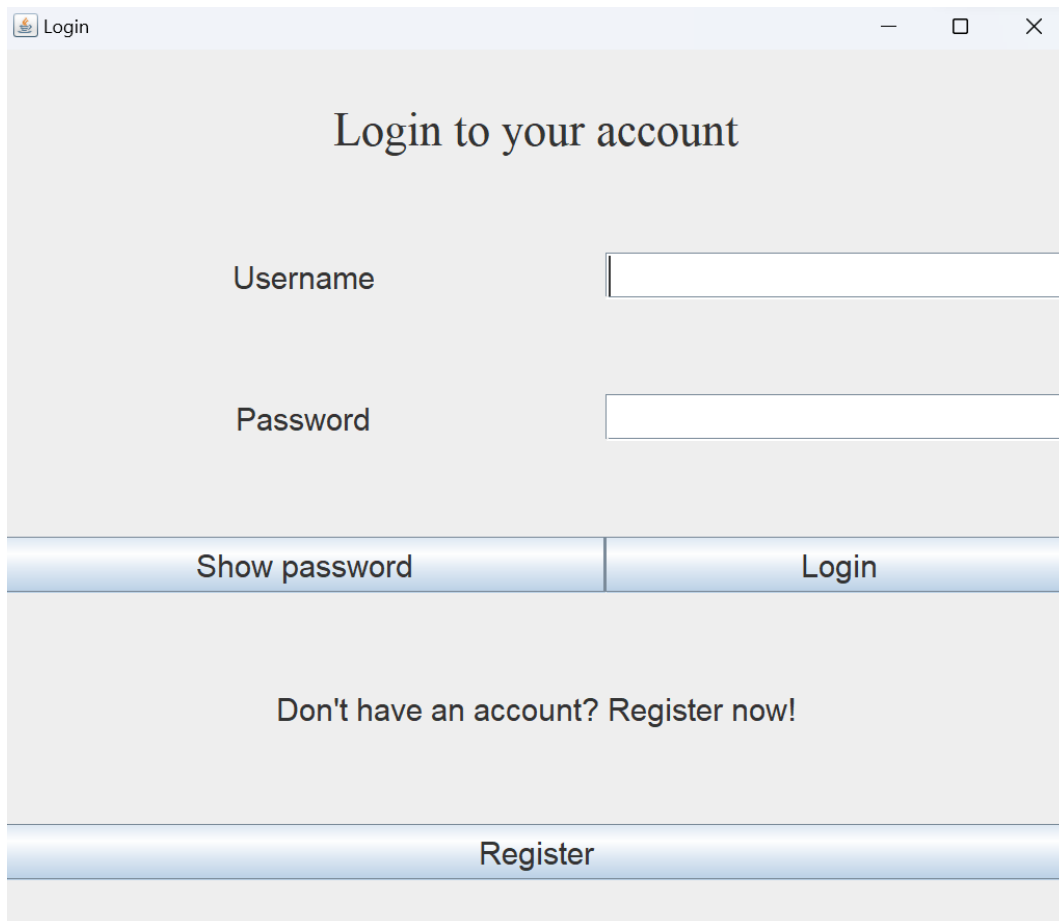
The Car table will store information about each car. Every car has a unique id, a brand, a model name, a color, and a type of fuel.

The Service Log table will store information about a car and its owner that is present in the shop. An owner can have multiple cars in the shop at once, but a unique car can be in the shop only once at a given time.

4. Implementation

4.1. Login window

The application was implemented using multiple View classes, which are created and made visible at dynamically at runtime, based on the actions of the user. The application starts with the login view, which presents 2 text fields and a button, along with some labels that prompt the user to enter a username and password in order to login. Along these, a register button is present for users to register. When the register button is pressed, the username and password present in the text fields are taken and a new user is created. A registration will automatically give the role of “employee” to the user. The login view is presented below.

The image shows a screenshot of a web application window titled "Login". The window has a light gray background. At the top, the title "Login to your account" is centered in a dark font. Below the title, there are two text input fields. The first field is labeled "Username" and the second is labeled "Password". Below the "Password" field, there are two buttons: "Show password" and "Login". At the bottom of the window, there is a link that says "Don't have an account? Register now!" and a "Register" button.

4.2. Employee window

If the user logs in as an employee, a new employee view is shown and the login view disappears. In this employee view, a table is presented which contains information about all cars and their owners that are present in the shop at the current time. The records of the shown table can be sorted by brand and/or fuel type and filtered by owner, car brand, fuel type and/or color. The combo boxes which allow the user to select what he wants to filter the table by are generated dynamically from the current cars and owners present in the database. Below the filtering options, there are 4 buttons: “owner operations”, “car operations”, “service operations” and “log out”. Clicking on any button from the first 3 will open a

pop-up window which will allow the user to perform Create, Read, Update and Delete actions on the table described by the name of the button. So, for example if the “owner operations” button is clicked, a pop-up window will open which allows the user to perform the operations already described above on the Owner table. The “search cars by owner” opens a pop up window with the cars that are owned by a certain owner. This window is opened whenever an option from the combo box on the right of the label is selected. The “Save car info in format” button saves the car information in the format specified in the combo box. By clicking the “log out” button, the user will be redirected to the login window once again. The employee window is presented below:

Employee window

Employee window

serviceNumber	carId	brand	modelName	color	fuelType	cnp	carOwner
1	1	car	car	car	car	2	suciu cezar
2	2	acar	acar	acar	acar	1	name surname
3	3	newcar	newcar	newcar	newcar	1	name surname

Sort car table by:
☐ Brand
☐ Fuel Type

Select filters to apply on the table above:
Owner
Car Brand
Fuel Type
Color

Search cars by owner:
Owner

Owner operations
Car operations

Service operations

Save car info in format
csv

Log out

The if the buttons “owner operations”, “car operations” and “service operations” are clicked, a operations window will be dynamically created. For example, the window generated by clicking “owner operations” contains a table that shows all the owners that are currently present in the database, as well as 3 buttons from which a new owner can be inserted, a selected owner can be edited or a selected owner can be deleted. If the first 2 buttons are clicked, a new pop-up window will be created and shown, where the user must enter values for each column of an owner, then press “submit”. The owner, car and service operations windows are presented below.

Owner operations

Owner operations

cnp	name	surname
2	suciu	cezar
3	name	nma

Add a new Owner

Update selected Owner

Delete selected Owner

Car operations

Car operations

id	brand	modelName	color	fuelType
1	car	car	car	car
2	acar	acar	acar	acar

Add a new Car

Update selected Car

Delete selected Car

Service operations

ServiceLog operations

serviceNu...	carId	brand	modelName	color	fuelType	cnp	carOwner
1	1	car	car	car	car	2	suciu cezar

Add a new ServiceLog

Update selected ServiceLog

Delete selected ServiceLog

4.3. Manager window

If the user logs in as a manager, a manager window will be generated which shows the same table, sorting options and filtering options as the ones in the employee window and a log out button which has the same functionality the other log out. Also, the search and save functionalities are the same as in the employee window. The manager window is presented below:

Manager window

Manager View

serviceNumber	carId	brand	modelName	color	fuelType	cnp	carOwner
1	1	car	car	car	car	2	suciu cezar
2	2	acar	acar	acar	acar	1	name surname
3	3	newcar	newcar	newcar	newcar	1	name surname

Sort car table by:

☐ Brand
 ☐ Fuel Type

Select filters to apply on the table above:

Owner

Car Brand

Fuel Type

Color

Search cars by owner:

Owner

Save car info in format

CSV

Log out

4.4. Admin window

The admin contains a table which shows the users that are present in the User table in the database, a combo box that filters the user table based on the role of the users, a button that takes the user to the Create, Read, Update and Delete operations on users and a log out button. If the “user operations” button is clicked, a pop-up window will open which contains a table with the current users and 3 buttons: “add a new user”, “update selected user”, “delete selected user”. If the first button is clicked, a new pop-up window will open where the current admin will have to complete the text fields corresponding to each column of the user table, then click “submit”. If the second button is clicked, the selected user from the table will be updated with the fields that the current admin will complete (the same procedure as in the insert operation). If the third button is clicked, the selected user from the table will be deleted. For each action performed, the table will refresh automatically both in the admin view and in the pop-up operations view. The admin window and the pop-up window that contains the user operations are presented below:

Admin window

Admin Window

id	username	password	role
1	admin	admin	admin
2	employee	employee	employee
3	manager	manager	manager

Filter users by:

Role

User operations

Log out

User operations

id	username	password	role
1	admin	admin	admin
2	employee	employee	employee
3	manager	manager	manager

Add a new User

Update selected User

Delete selected User