

# Software design

## Documentation

### Project

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 .....	5
2.2.3. Sergeds library.....	5
2.2.4. Jackson Dataformat XML.....	5
2.2.5. Gradle .....	5
2.2.6. JUnit, Groovy and Spock .....	5
3. UML diagrams .....	6
3.1. Use case diagram .....	6
3.2. Class diagram .....	8
3.3. Entity-Relationship diagram.....	10
3.4. Activity diagrams.....	11
3.4.1. Activity diagram to change the language of the app .....	11
3.4.2. Activity diagram to login the app .....	11
3.4.3. Activity diagram to view the cars in service.....	13
3.4.4. Activity diagram to filter the table containing the cars in service .....	14
3.4.5. Activity diagram to search cars by owner .....	15
3.4.6. Activity diagram to save car information in different formats .....	16
3.4.7. Activity diagram to perform CRUD operations on tables.....	<b>Error! Bookmark not defined.</b>
3.4.8. Activity diagram to view car statistics .....	22
4. Implementation .....	42
4.1. Login window .....	42
4.2. Employee window.....	43
4.3. Manager window .....	46
4.4. Admin window .....	47

# 1. Objective

The objective of this project is to familiarize with the Client/Server architectural pattern, service-oriented architecture (SOA) patterns, and design patterns. A relational database (SQL Server, MySQL, etc.) will be used for data persistence.

## 1.1. General requirement

The general requirement for the first homework is:

Transform the application implemented in assignment 3 into a client/server application, using at least 5 design patterns (at least one creational design pattern, one behavioral design pattern, and one structural design pattern) and a service-oriented architecture (SOA).

- ❖ During the analysis phase, create a use case diagram and activity diagrams for each use case.
- ❖ During the design phase, create:
  - 2 class diagrams for the server application and the client application, adhering to the GRASP and DDD principles, using a service-oriented architecture (SOA) and at least 5 design patterns.
  - An entity-relationship diagram corresponding to the database.
  - Sequence diagrams corresponding to all use cases.
- ❖ During the implementation phase, write code to fulfill all the functionalities specified by the use case diagram, using:
  - The design provided by the class diagrams and sequence diagrams.
  - One of the following programming languages: C#, C++, Java, Python.
- ❖ The completion of the assignment will involve submitting a directory that includes:
  - A file containing the UML diagrams created.
  - The database.
  - The software application.
  - Documentation (minimum 20 pages) - a file that includes:
    - Student's name, group.
    - Problem statement.
    - Tools used.
    - 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 a client/server 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 the list of all existing vehicles in the service, sorted by brand and fuel type;
- ❖ Filter vehicles by 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 the list of all existing vehicles in the service, sorted by brand and fuel type;
- ❖ Filter vehicles by 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;
- ❖ View statistics related to vehicles using graphs (radial structure, ring structure, column structure, etc.).

After authentication, administrator users can perform the following operations:

- ❖ CRUD operations for information related to application users;
- ❖ View the list of all users and filter it by user type.
- ❖ Notify each user through at least 2 options (email, SMS, WhatsApp, Skype, etc.) about any changes in their authentication information.

**The graphical interface of the application will be available in at least 3 languages of international circulation.**

## 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
- Jackson Dataformat XML – library for deserializing XML files as POJOs
- Sun mail – library that allows sending emails
- Twilio - library that allows sending text messages
- 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. Jackson Dataformat XML

I used this external library in order to deserialize XML files into POJOs, which were further used for implementing the internationalization feature of the app. The app contains 3 XML files, each one containing every phrase used in all of the views. When a language is selected, the XML corresponding to that language is deserialized in a POJO, which is further used to dynamically set the labels and buttons text to the phrases translated in the selected language.

### 2.2.5. Sun mail library

I used this external library to facilitate the notification feature via email to the user when its account details have been changed. The email is sent via Gmail smtp and it requires authentication with a Gmail account. For this reason, I used my personal email and an “App password”, which was taken from my Google account, to authenticate and send emails from my email address to the user’s email address.

### 2.2.6. Twilio library

I used this external library to facilitate the notification feature via WhatsApp to the user when its account details have been changed. The Twilio library contains an API which allows a user to send WhatsApp messages and text messages on behalf of the application using the Twilio phone number/account. To connect to the API, an account\_sid and an authentication token are required, which are granted to Twilio users when they register on the website.

### 2.2.7. 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 to create an opinion on which one I would prefer to use day by day.

### 2.2.8. 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 diagrams, the class diagrams, and the entity-relationship diagram had to be created.

#### 3.1. Use case diagrams

The first one was the use case diagrams, which were created during the analysis phase of the project. Because the project contains both a server and a client service, two use case diagrams had to be created, each corresponding to a service. The first one is the client service use case diagram (Figure 1).

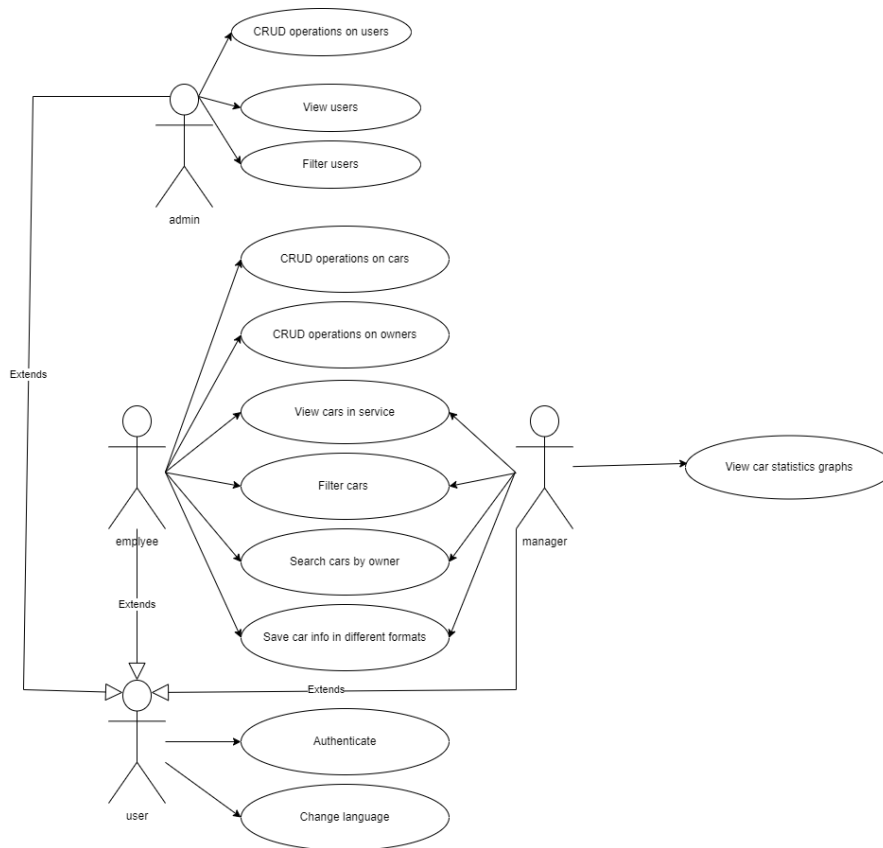


Figure 1

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

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, 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 can 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 can 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. The manager can also view statistical graphs about the cars that are present in the service shop.

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.

The second use case diagram corresponds to the server service (Figure 2).

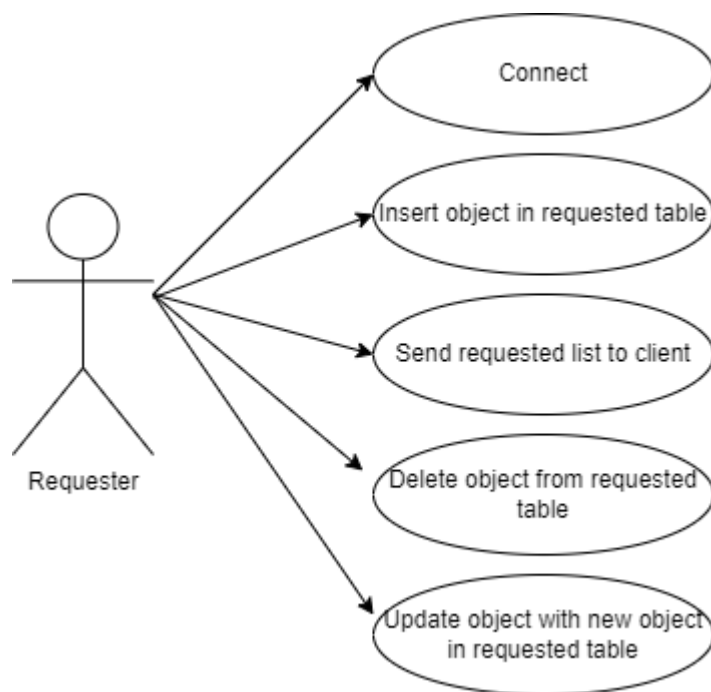


Figure 2

The server can perform several actions: connect to a requester, inset a given object in a table that is requested, send a requested list to the requester, delete a given object from the table the object belongs to, update an object and with the fields of another object from the table it belongs to. All these actions are performed whenever a "Requester" asks the server to perform them.

### 3.2. Class diagrams

During the design phase of the project, class diagrams were created which show the classes of the application and the relationship between them. Because the application is split into two services, two diagrams were created: one for the client service and one for the server service. The first diagram corresponds to the client service side classes (Figure 3).

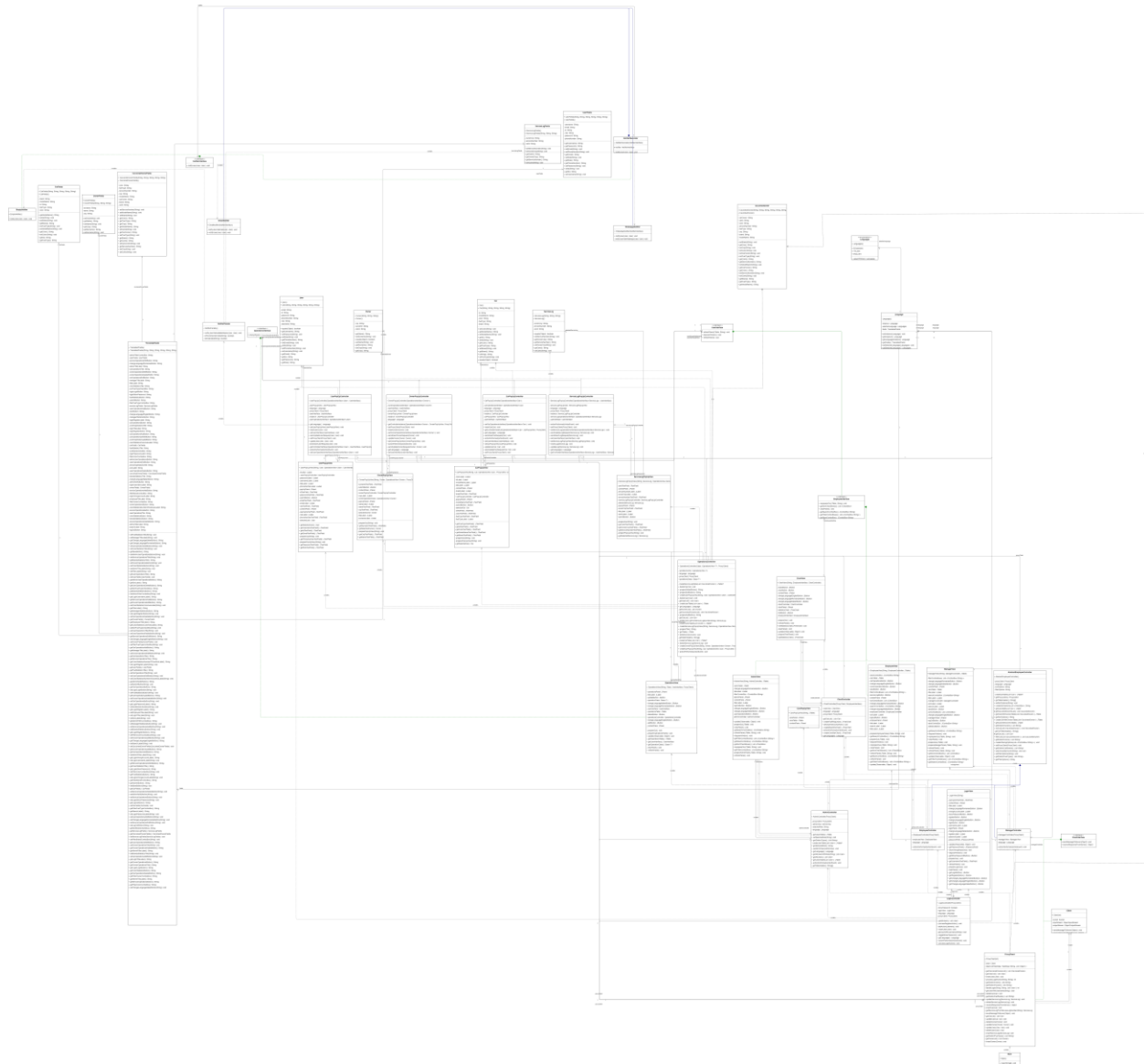


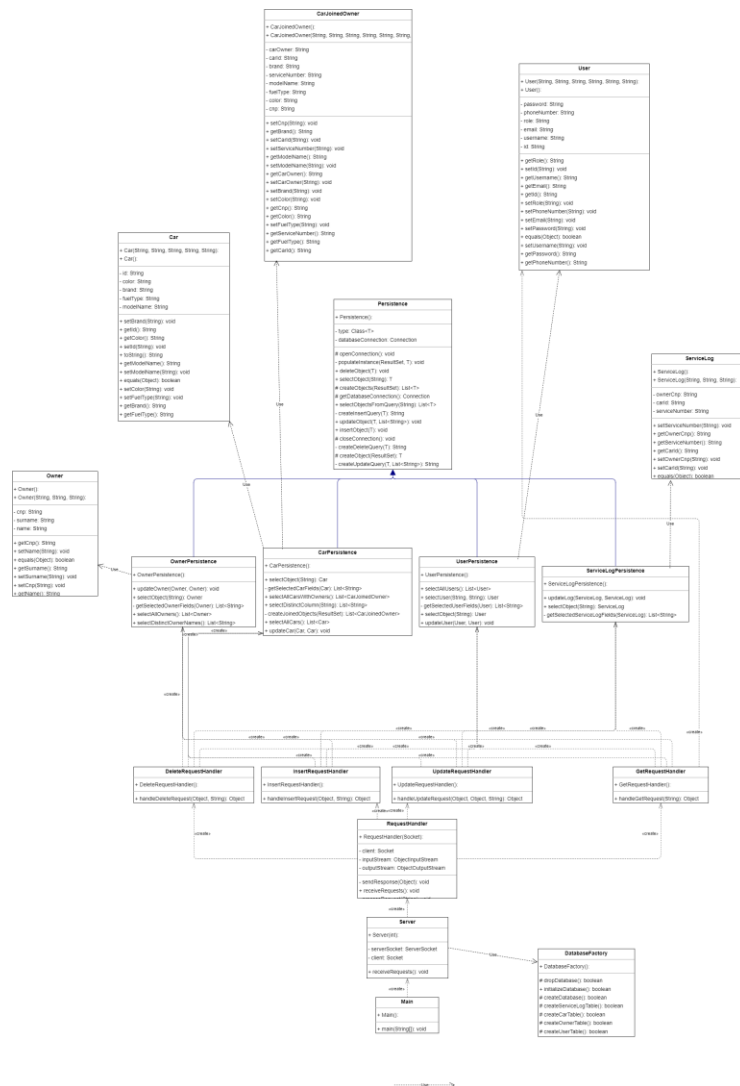
Figure 3

In this class diagram we can see that the MVC architectural pattern was respected, each class being part of one of the following packages: model, persistence, view or controller. 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 in the view's respective controller that, when triggered, will perform a specific action such as login or CRUD operations on a table. Each view has a controller associated to it that acts



Furthermore, the Domain Driven Design was respected, each class being part of one of two domains: the user domain and the car domain. Thus, every class that can be categorized as a user domain class was placed in the first domain. These classes include the login/employee/manager/administrator views and their respective controllers, the user model class and the user pop-up window and its respective controller. The other classes were placed inside the car domain.

Figure 4



The server service contains the classes that are responsible for creating and managing the database, as well as the classes that are responsible for handling requests that are received from the client that connected to the server.

### 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 (Figure 5).

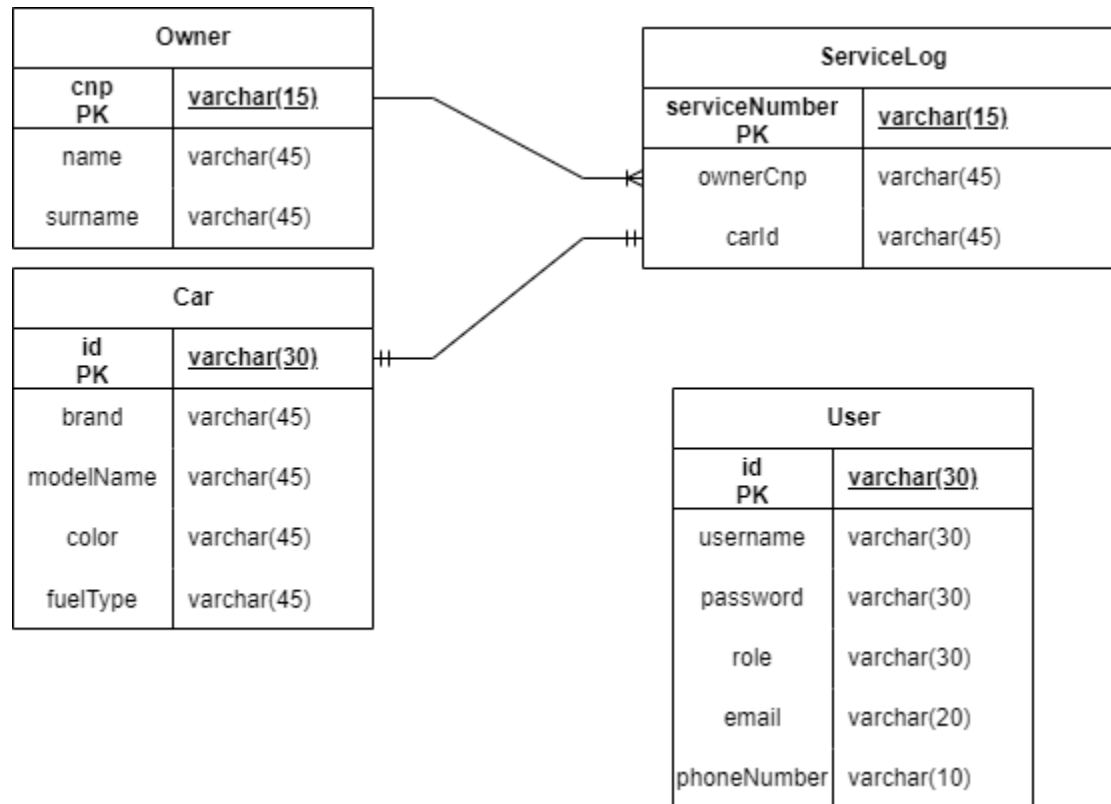


Figure 5

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 email and phone number columns store two strings, which are used to notify the user whenever it's details are changed (if the email/phone number are valid).

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.

### 3.4. Activity diagrams

The activity diagrams are used alongside the use case diagram to provide a more detailed explanation for each use case presented in the use case diagram. Thus, the following activity diagrams were created.

As there are two use case diagrams, the activity diagrams can be split into two categories: the activity diagram for the client service and the activity diagram for the server service.

#### 3.4.1. Activity diagrams for the client service

##### 3.4.1.1. Activity diagram to change the language of the app

The following activity diagram presents the steps a user has to take in order to change the language of the application (Figure 7).

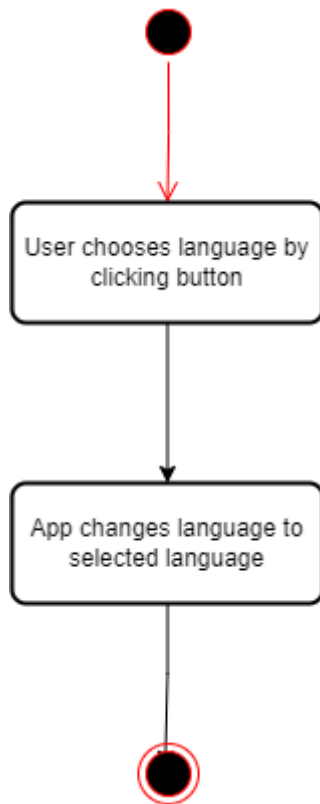


Figure 6

Since the user can change the language in the same way in every window of the application, the diagram contains only two steps. First, the user must click on one of the 2 available buttons that are responsible with changing the language of the app to a different one. These 2 buttons change based on the selected language. So, if the application is currently in English, the two available buttons are "Romanian" and "Italian". If the application is currently in Romanian, the two available buttons are "English" and "Italian". If the application is currently in Italian, the two available buttons are "English" and "Romanian". The second step is the app changing the language to the one that the user has selected.

##### 3.4.1.2. Activity diagram to log in the app

The following diagram presents the steps a user must perform in order to successfully log in the application (Figure 8).

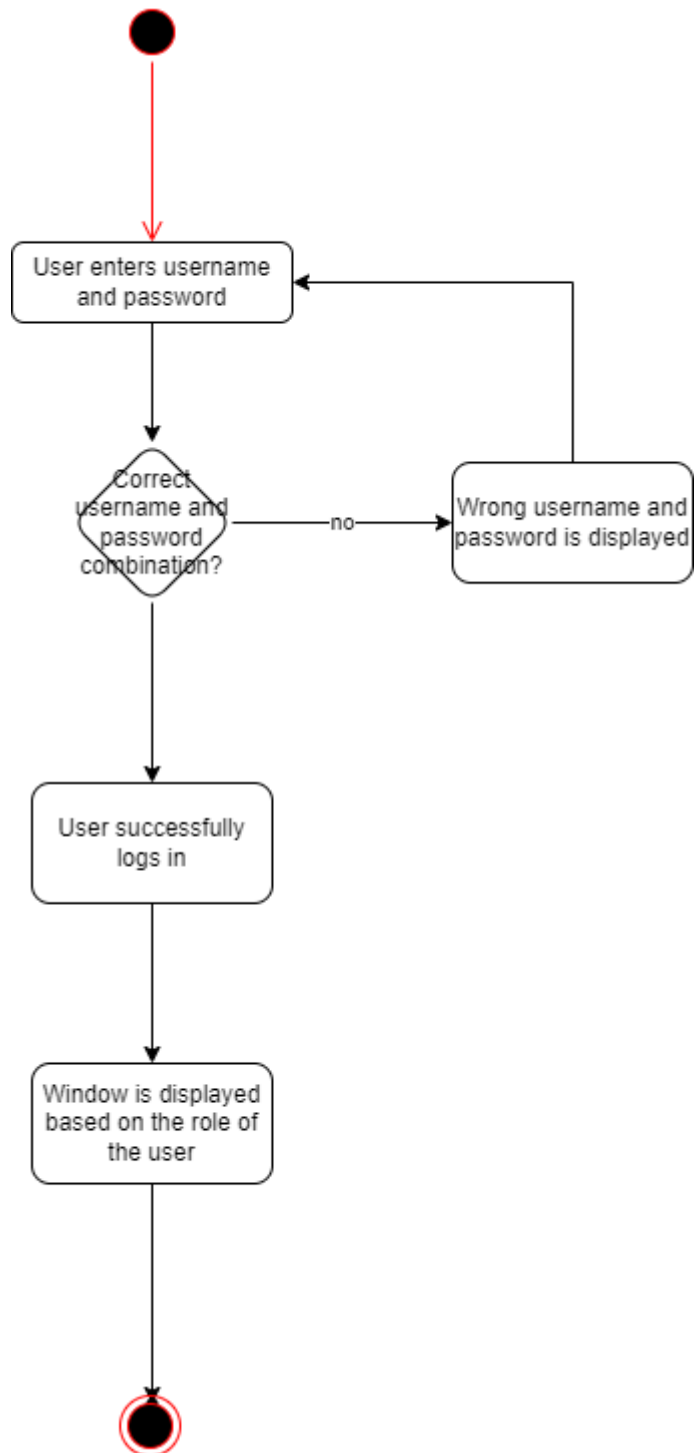


Figure 7

This activity diagram contains the following steps: First, the user must enter a username and password. Then, if the username and password match an account in the database, the user is logged in and a new window is displayed based on the role of the user. The roles can be employee, manager, or administrator. If the username and password combination don't match an account in the database, the user is not

successfully logged in and a text will be displayed that informs the user that he introduced a wrong username and password combination.

#### 3.4.1.3. Activity diagram to view the cars in service

The following diagram represents the steps that the user must take to see what cars are in service and takes into account that only managers and employees can see the cars in service, so it only applies if the user is logged into an account with the role “employee” or “manager” (Figure 9).

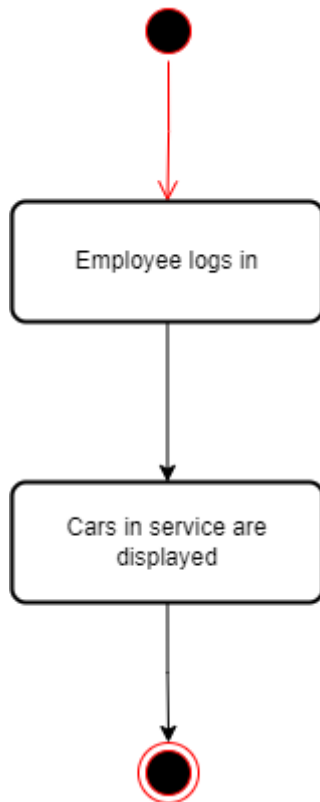


Figure 8

The diagram for the manager is the same as the one shown above, but instead of the employee log in it is manager log in. The steps that are necessary to be taken to see the cars in service are the log in, the table being shown on the window that is shown right after the log in.

#### 3.4.1.4. Activity diagram to filter the table containing the cars in service

The following activity diagram is used to represent the steps that must be taken for the employee/manager to filter the cars in service table (Figure 10).

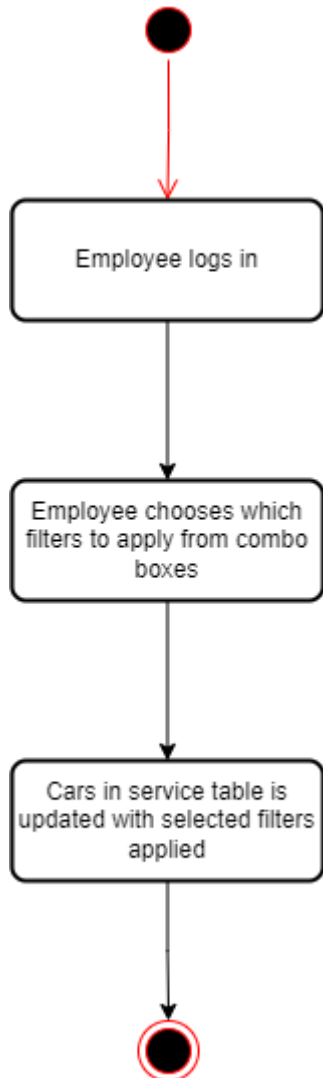


Figure 9

The activity diagram for the manager is the same as the one displayed above. The user must log in either as an employee or as a manager and select which filters to apply on the table from the given combo boxes. The table is then updated with the information containing only the rows that satisfy the given filters.

#### 3.4.1.5. Activity diagram to search cars by owner

The following diagram is used to show the steps that an employee or manager must take in order to search all the cars that a person has in the shop (Figure 11).

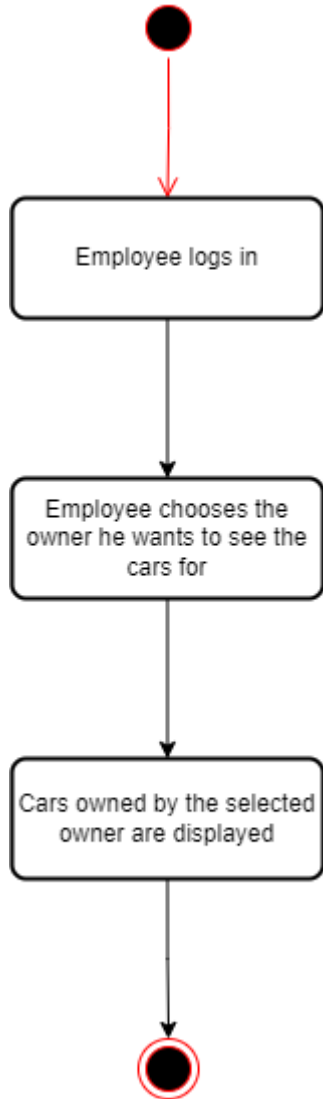


Figure 10

The activity diagram for the manager is the same as the one displayed above. The steps that need to be accomplished are the same as in the case of the filtering activity diagram, except rather than selecting which filters to apply, the user must select the owner he wants to see the car info for.

#### 3.4.1.6. Activity diagram to save car information in different formats

The activity diagram to save the car information is the same for both employee and manager user types (Figure 12).

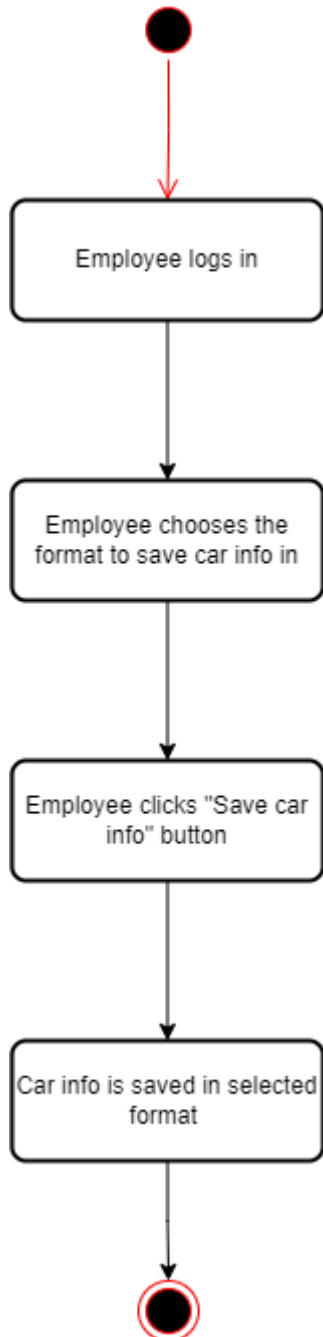


Figure 11

The user must first log in, then select which format he wants the car information to be saved in, then he must click the button to perform the information save action.



#### 3.4.1.7. Activity diagram to insert a new user

To perform operations on the tables Car, Owner and Service Log, the user must be an employee.

To perform operations on the User table, the user must be an administrator.

The following activity diagram shows the steps that are taken to insert a new user. This activity diagram applies to all the other tables as well (car, owner, service log). The diagram is shown in Figure 12.

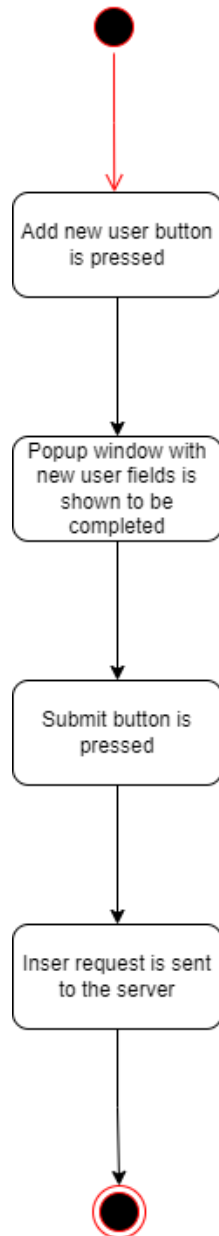


Figure 12

The previous activity diagram is also available for the insert operations on car, owner and service log. As shown, in order to insert a new user, a request must be made towards the server, which must fulfill it.

#### 3.4.1.8. Activity diagram to delete a user

The restrictions of operations on tables remain the same as the ones in the insert activity diagram. The following activity diagram shows the steps that are taken to perform a deletion of a user (Figure 13).

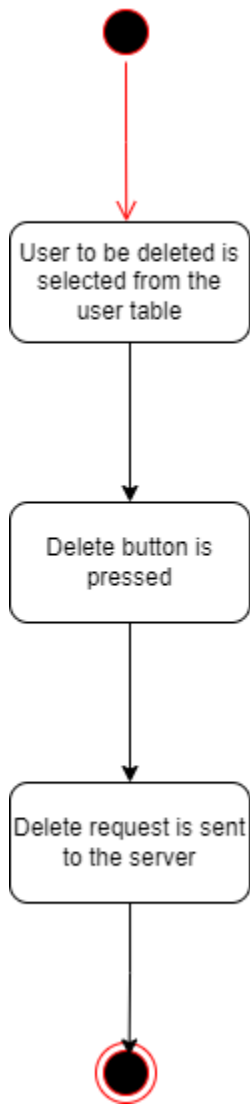


Figure 13

This activity diagram also applies for the Car, Owner and Service Log tables. As also shown in the insert activity diagram, the deletion is not done in the client service. The client simply request the deletion to be made towards the server service.

#### 3.4.1.9. Activity diagram to read users

The restrictions from the previous two also apply here. The following activity diagram shows the steps that are taken to read users from the database (Figure 14).

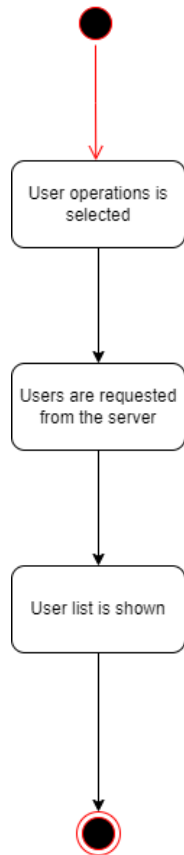


Figure 14

This activity diagram also applies to the Car, Owner and Service Log tables. As shown, in order to read users from the database, the client must request towards the server the list of users and when the list is received it is shown in the view.

#### 3.4.1.10. Activity diagram to update users

The previous restrictions apply here as well. Figure 15 shows the steps that are taken to update a user.

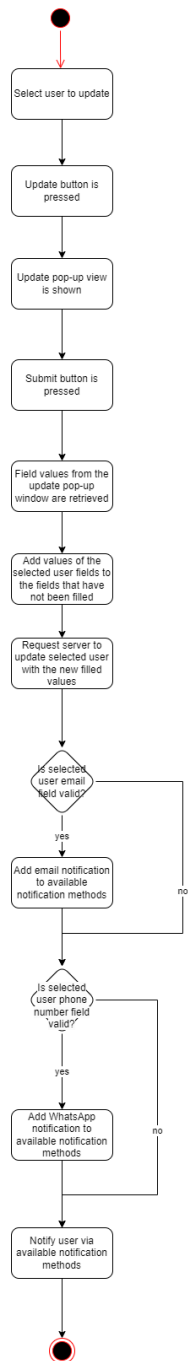


Figure 15

This activity diagram is available only for the User table, because it also contains the notification of the user in case it has a valid email address or phone number.

#### 3.4.1.11. Activity diagram to update an object

The following activity diagram (Figure 16) presents the steps that are taken when updating an object that is not a user. So, it includes owners, cars and service logs.

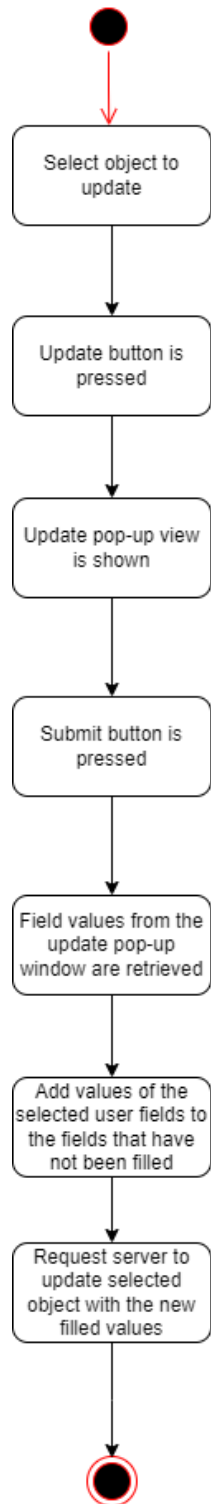


Figure 16

The difference between Figure 16 and Figure 15 is the fact that in Figure 16 there are no notification steps. So, only when updating a user, the user is notified of the change.

#### 3.4.1.12. Activity diagram to view car statistics

The following activity diagram shows the necessary steps a manger needs to take in order to see statistics about cars (Figure 17).

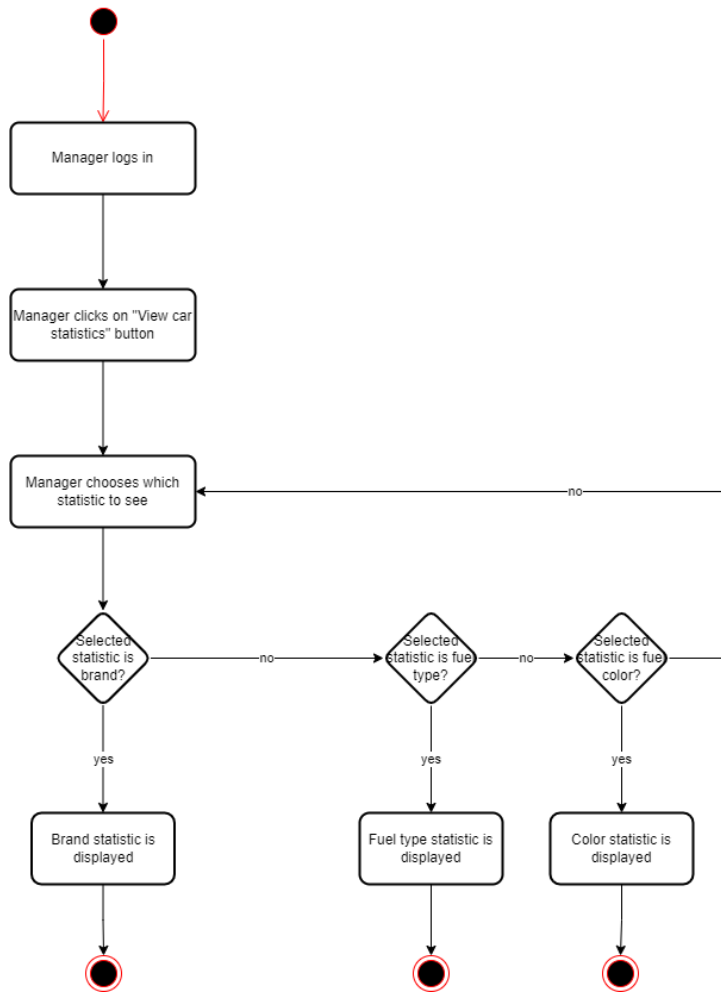


Figure 17

There are 3 available statistics: brand, type and color. To show each one, a manager must log in, click on the "statistics" button, choose which statistic he wants to see by pressing one of the 3 available buttons and based on the clicked button, the selected statistic is displayed.

#### 3.4.2. Activity diagrams for the server service

The server service provides functionalities such as connecting to a requester, inserting an object in a table, updating/deleting an object from a table, and sending lists of objects to the requester. These functionalities all have an activity diagram which will be presented below.

##### 3.4.2.1. Activity diagram to connect to a requester

Figure 18 shows the steps that are taken to connect the server service to a requester.

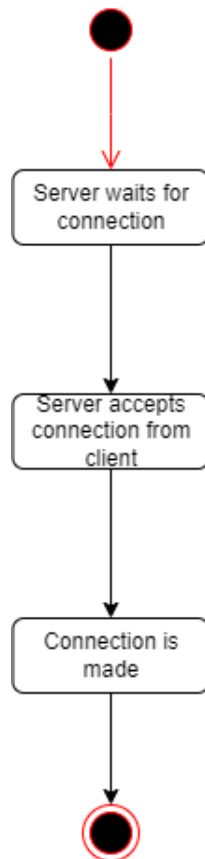


Figure 18

First, the server starts to wait for connections, which blocks the execution of the server until a connection request is made. When the connection request is made, the server accepts the requests and establishes the connection.

#### 3.4.2.2. Activity diagram to insert an object in a table

The following activity diagram shows the steps that are taken to insert an given object in a table (Figure 19).

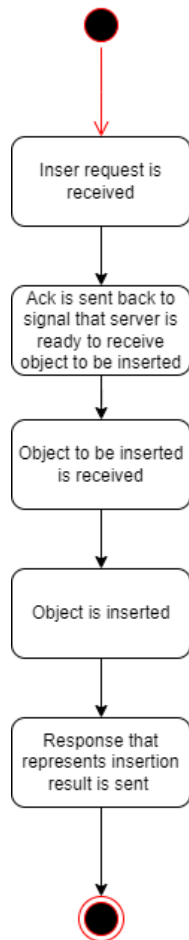


Figure 19

The insert process is done in several steps: first the insert request is received, then the server sends an acknowledgement back to the requester to signal that it is ready to receive the inserted object, then the inserted object is received and inserted in its corresponding table. Lastly, a response is sent to the requester with the result of the insertion. This activity diagram applies to all tables: User, Owner, Service Log, Car.

#### 3.4.2.3. Activity diagram to update an object in a table with another object

The following activity diagram shows the steps that are taken to update an given object in a table with another object (Figure 20).



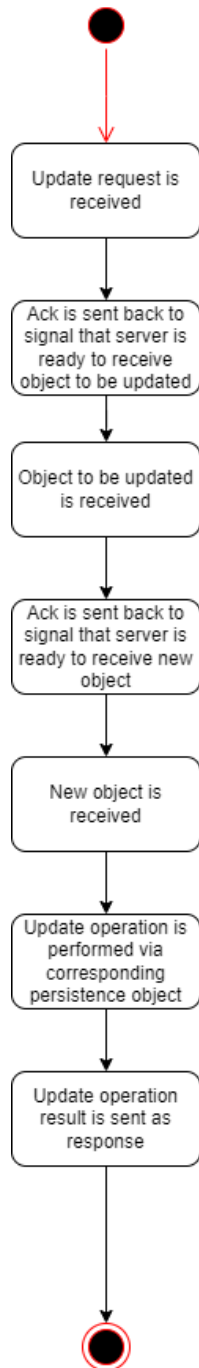


Figure 20

The steps that are taken to update an object with another object are similar to the ones that are taken to insert a new object. However, 2 acknowledgements and 2 objects are received in stead of 1 acknowledgement and 1 received object like in the case of an insert.

#### 3.4.2.4. Activity diagram to delete an object from a table

The following activity diagram shows the steps that are taken to delete an given object from a table (Figure 21).

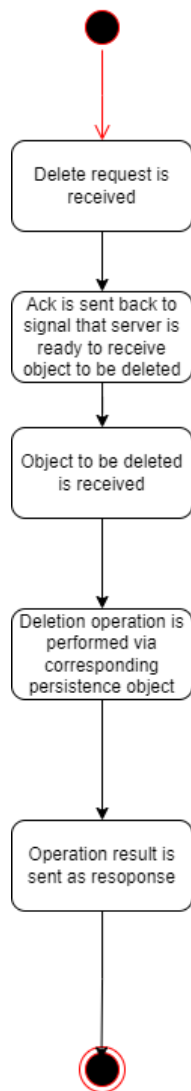


Figure 21

The steps that are taken to delete an object from a table are the same as the ones that are taken to insert an object in a table (Figure 19).

#### 3.4.2.5. Activity diagram to send a requested list to the requester

The following activity diagram shows the steps that are taken to send a list object to the requester (Figure 22).

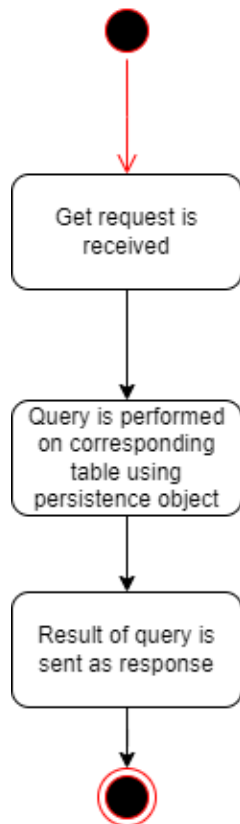


Figure 22

The steps that are taken to send a list of objects to the requester are: first the get request is received, then the corresponding query on the requested table is performed, then the result is sent as a response to the requester.

### 3.5. Sequence diagrams

The sequence diagrams detail the activity diagrams for each use case present in the use case diagram. As in the activity diagrams case, the sequence diagrams can be split into two categories: the sequence diagrams for the client service and the sequence diagrams for server service. The sequence diagrams for CRUD operations are the same for all tables: User, Car, Owner, Service Log. Thus, only one sequence diagram is shown for each operation.

### 3.5.1. Sequence diagrams for the client service

#### 3.5.1.1. Sequence diagram for authentication

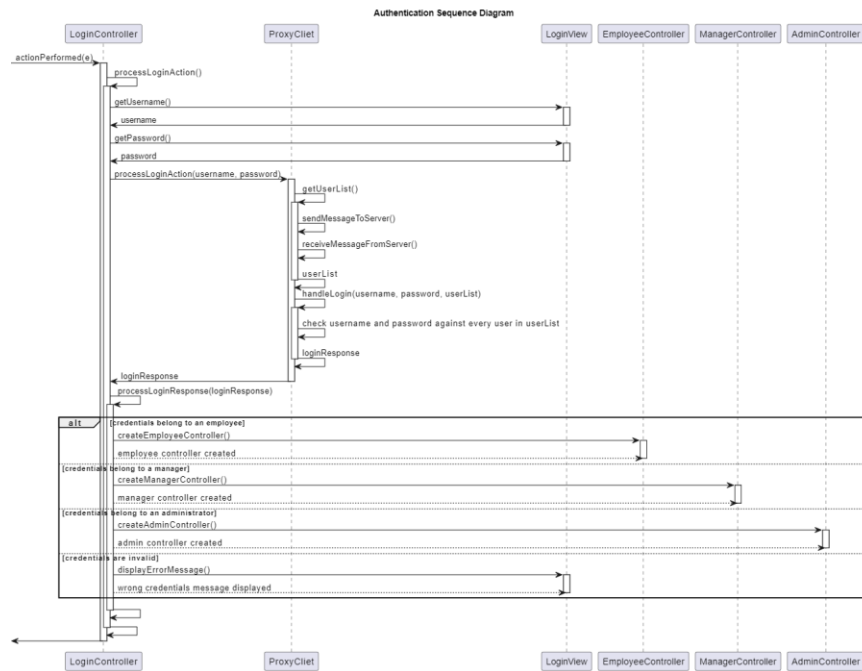


Figure 23

For the authentication, the username and password are taken from the login view into the login controller, then the list of users present in the database is taken from the server using the proxy client. Based on whether the credentials are valid or not, the user logs in as an employee, a manager or an administrator. If the credentials are not valid, a message is displayed.

3.5.1.2. Sequence diagram for changing the language

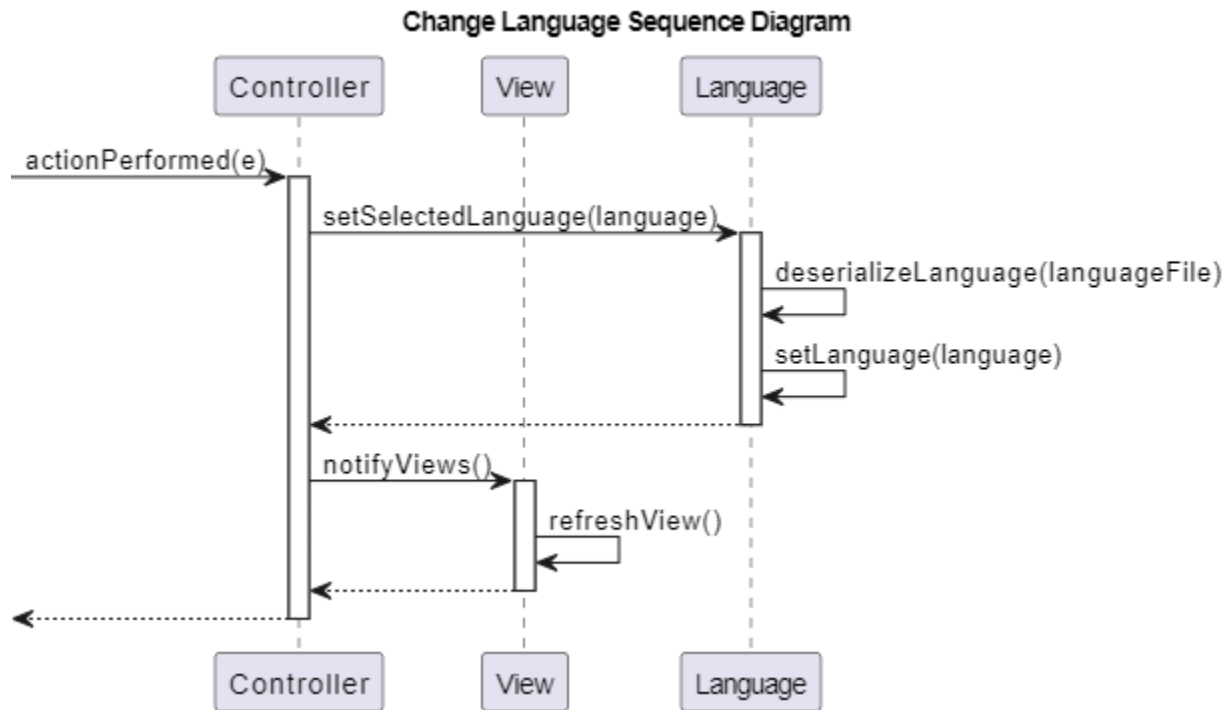


Figure 24

Whenever a command to change the language is received, the selected language is updated, the file corresponding to the new language is deserialized and the views are refreshed and rendered dynamically based on the current selected language.

### 3.5.1.3. Sequence diagram for viewing users

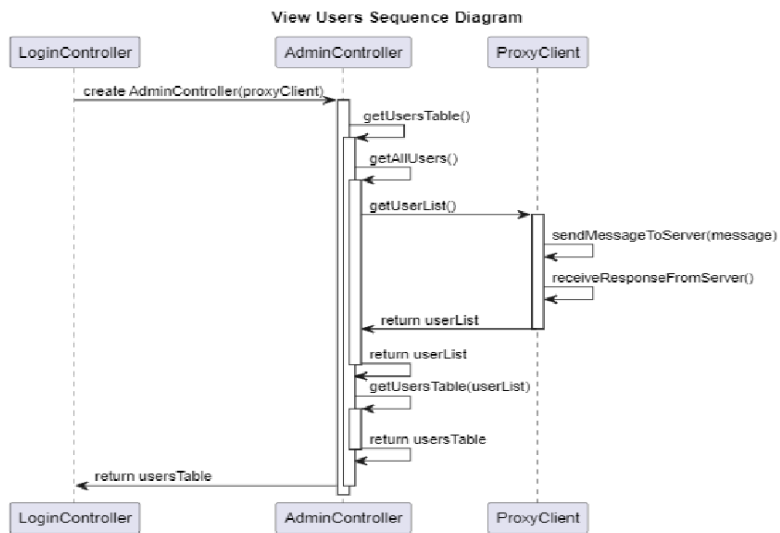


Figure 25

To view the users in the database, the proxy must first request the list from the server. Then, after the list is received, it can be rendered into a table.

#### 3.5.1.4. Sequence diagram for inserting a user

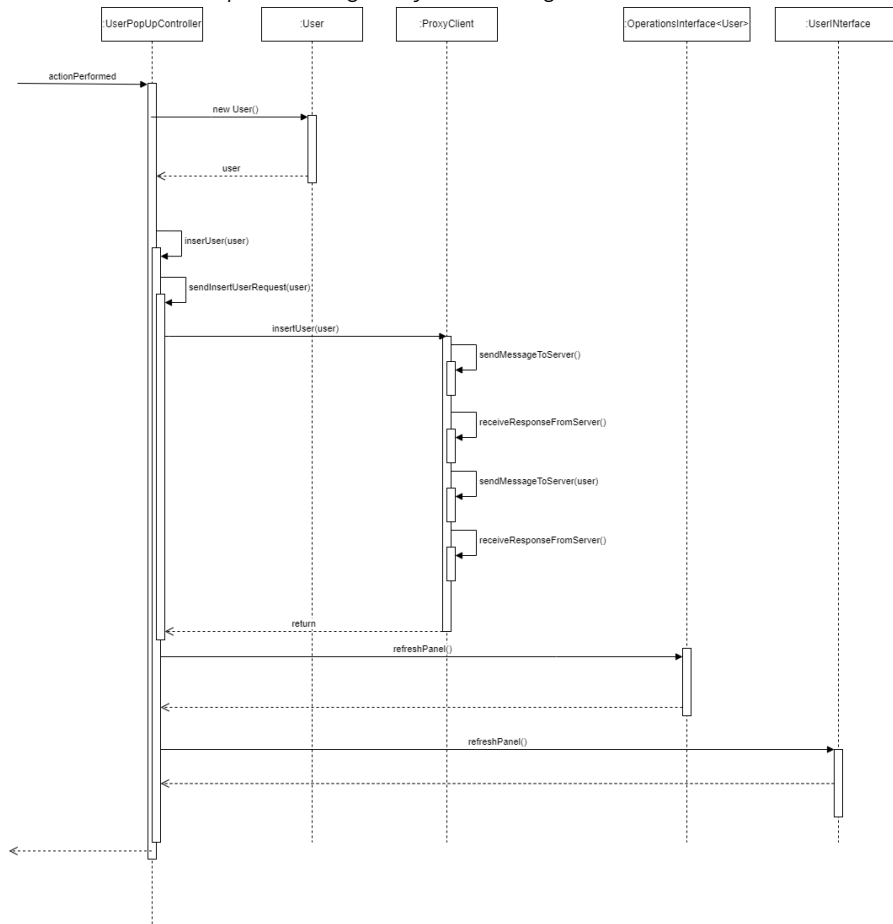


Figure 26

To insert a user in a table, the user to be inserted is first created by taking the values from the text fields that are present in the current view, then the proxy sends a request to the server to insert the given user into the database. Finally, the views are refreshed to display the live information.

### 3.5.1.5. Sequence diagram for updating a user

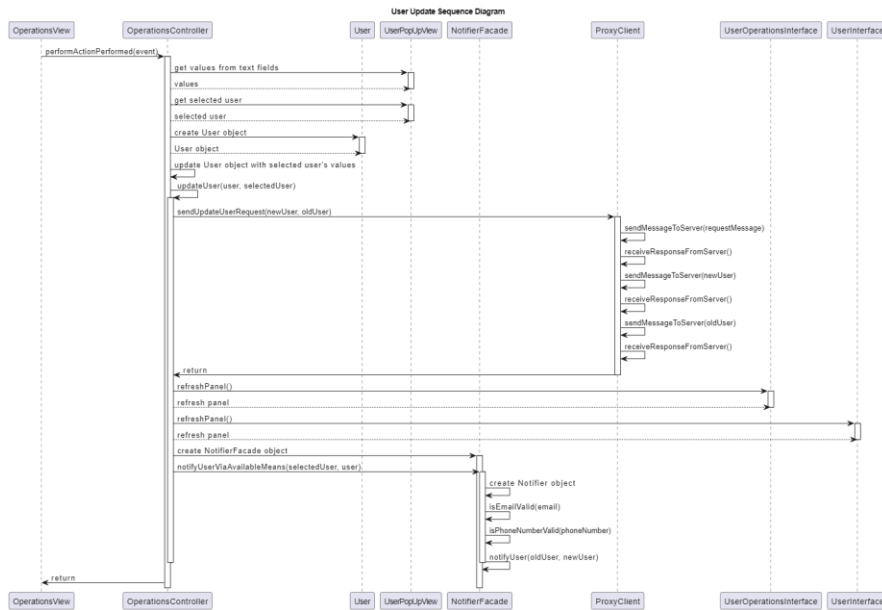


Figure 27

The steps required to update a user are like inserting one. After the update request is fulfilled by the server, the user is notified that its account details have been changed via his old available email address and phone number (if they are valid).

### 3.5.1.6. Sequence diagram for deleting a user

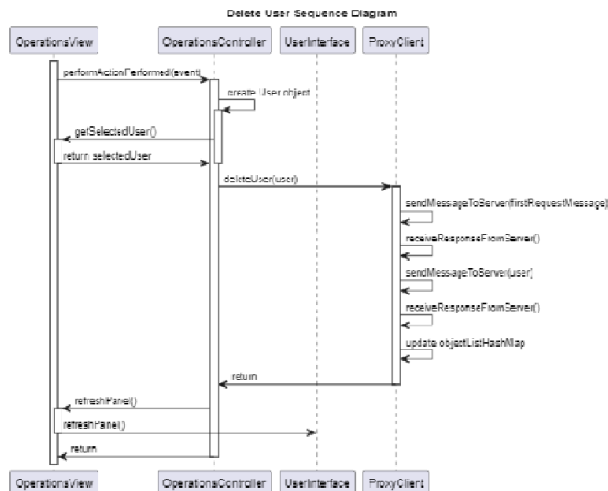


Figure 28

To delete a user, the steps are the same as the ones present in the insertion sequence diagram.



### 3.5.1.7. Sequence diagram for filtering users

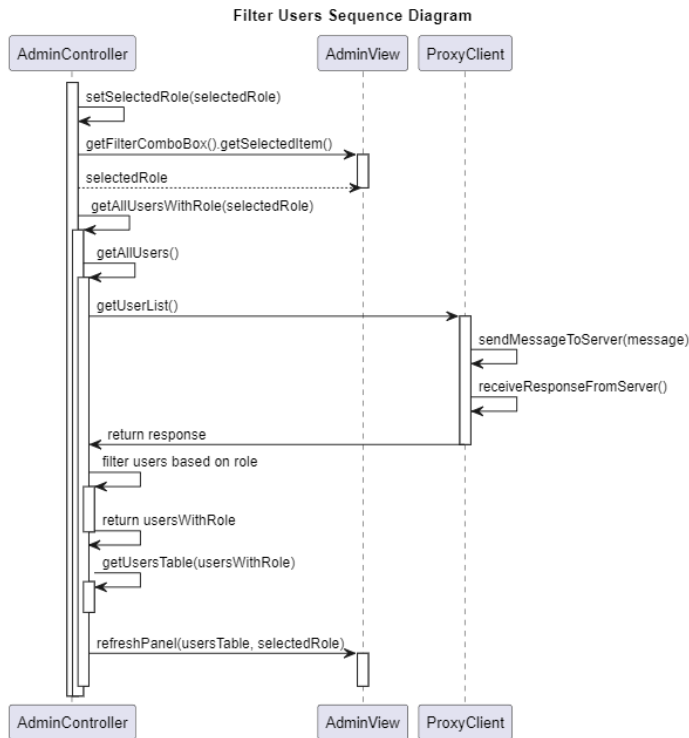


Figure 29

To filter users after a given filter, the admin controller gets the selected role which acts as the filter, gets all the users present in the database via the proxy, then filters that list against the selected role and creates a table out of the resulting list. Finally, the view is updated to show the filtered users in the table.

### 3.5.1.8. Sequence diagram for searching cars by owner

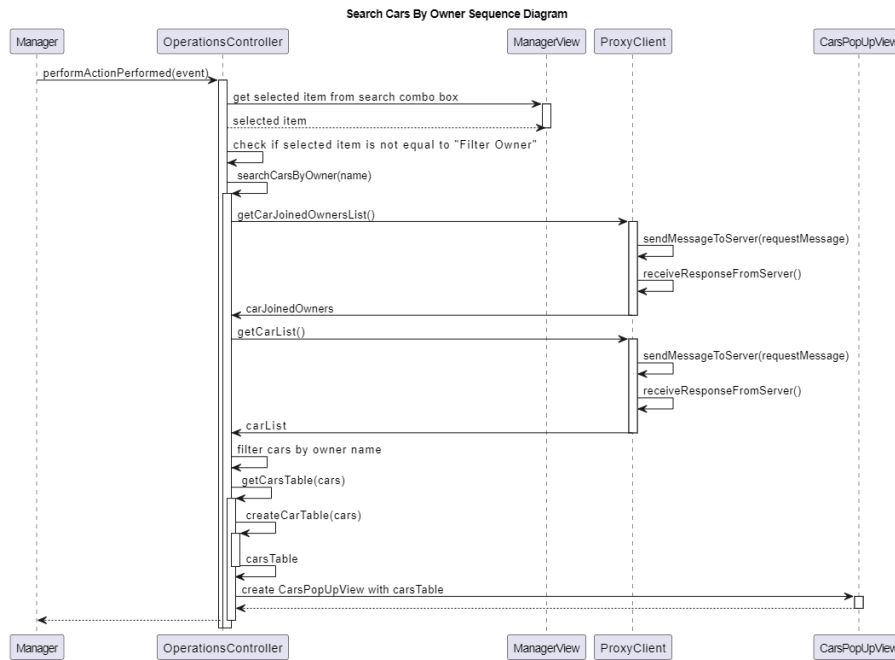


Figure 30

To search for cars by owner name, the owner's name is taken from the combo box present in the view, then the list of all cars with owners and the list of all cars are retrieved from the server via the proxy. The received list of cars with owners is then filtered to contain only the id of the cars which the owner possesses, and the car list is filtered to contain only the cars whose id is present in the car id list. Finally, a pop-up view is created to display the table with the cars.

3.5.1.9. Sequence diagram for saving car information in different formats

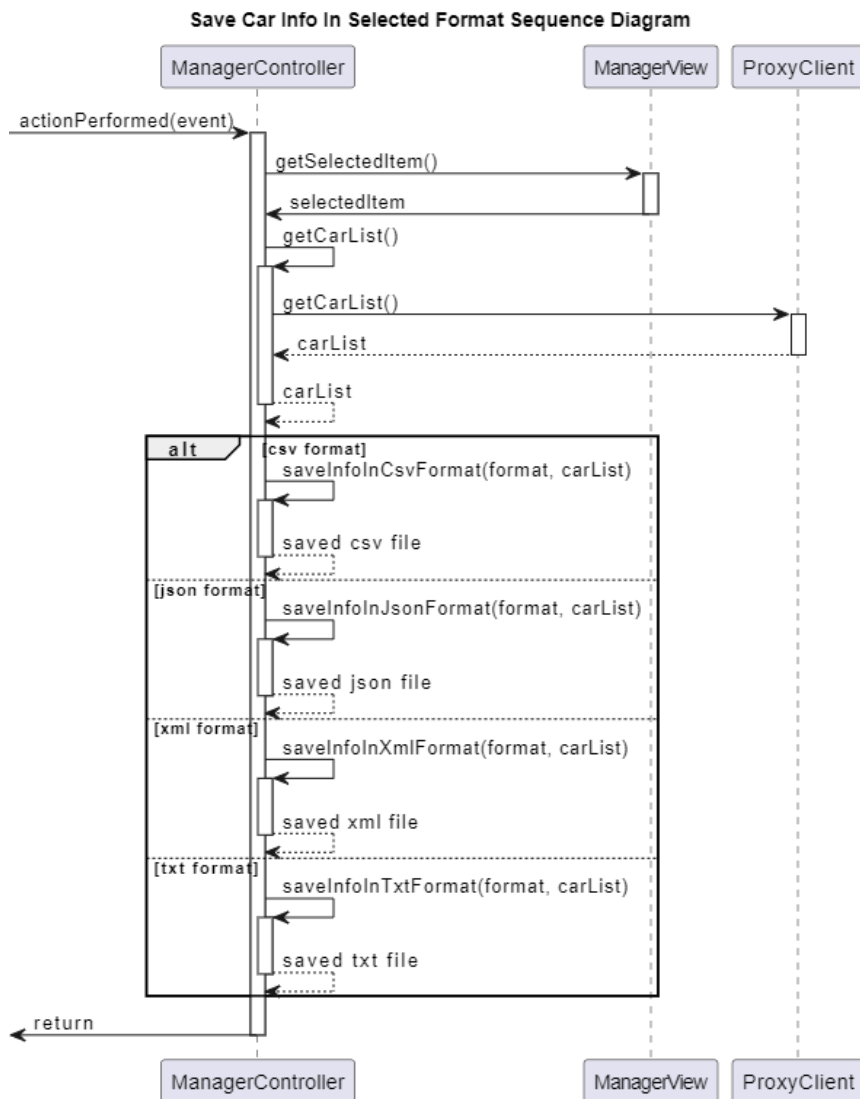


Figure 31

To save the car table information in a selected format, the car list is requested from the server via proxy client, then it is saved in the selected format.

### 3.5.1.10. Sequence diagram for viewing car statistics

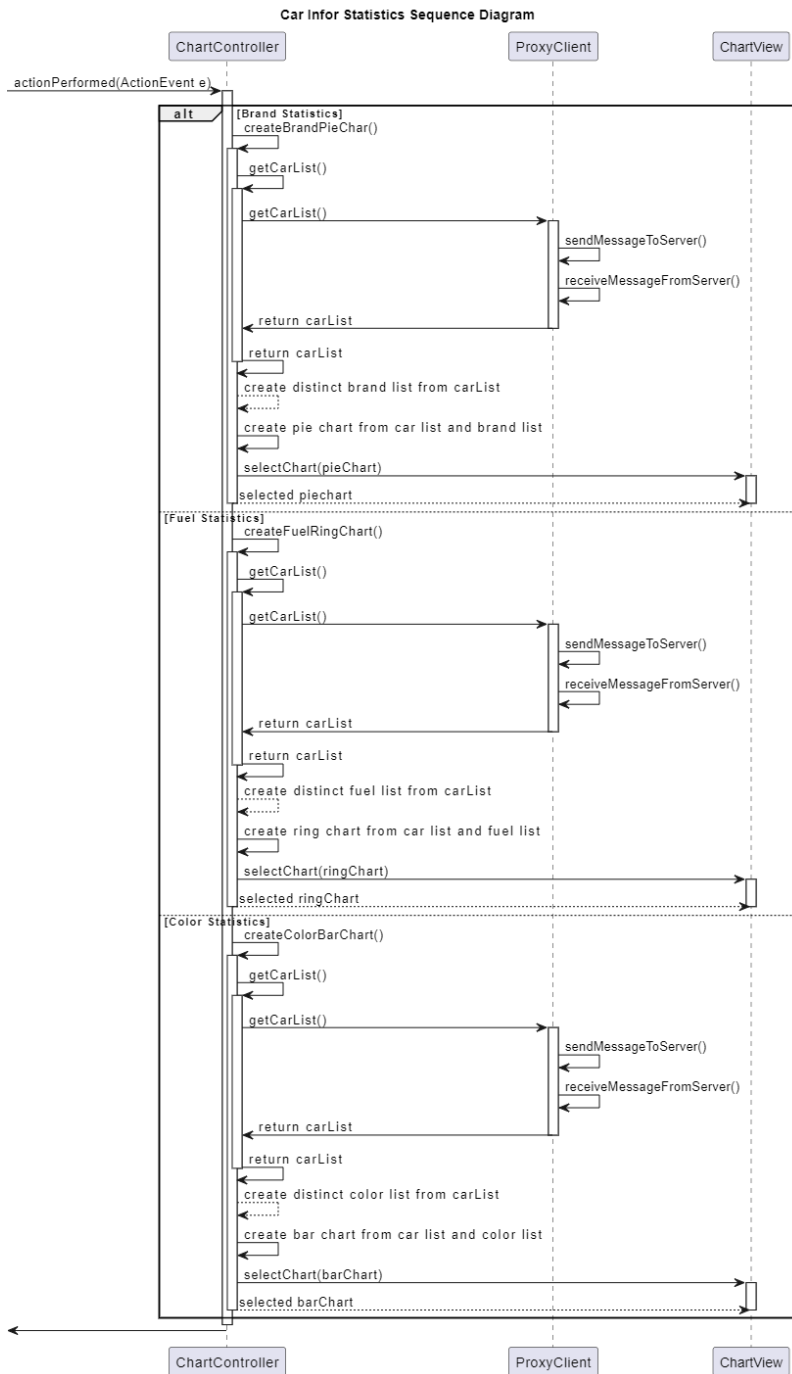


Figure 32

To view car statistics, the car list is requested from the server via the proxy client and based on which chart is selected to be shown, a pie, ring or bar chart is created and displayed.

### 3.5.2. Sequence diagrams for the server service

#### 3.5.2.1. Sequence diagram for connecting the server to a requester

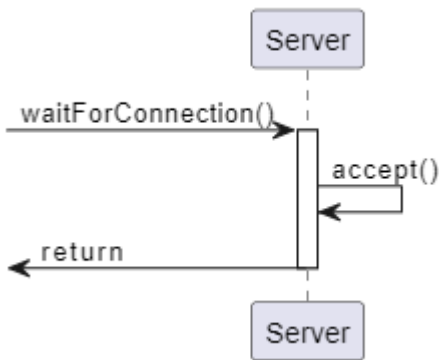


Figure 33

To connect the server service to a requester, the server starts to wait for connections by using the `accept()` method, which blocks the execution of the server until a connection is established.

#### 3.5.2.2. Sequence diagram for inserting a given object in its corresponding table

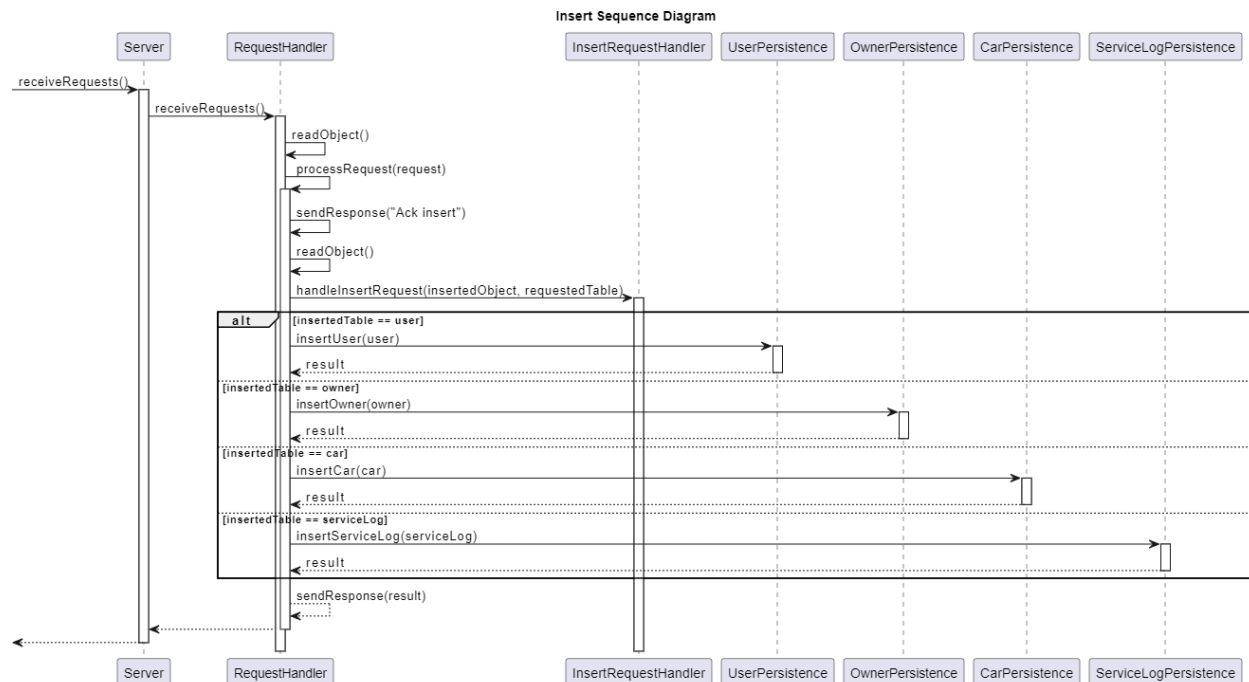


Figure 34

To insert a new object in its corresponding table, an insert request is received from the requester, an ack is sent back to signal that the server is ready to receive the object to be inserted, the object is received and based on its class, its corresponding persistence object is created to execute the operation of insertion in the correct table.

### 3.5.2.3. Sequence diagram for updating a given object in its corresponding table

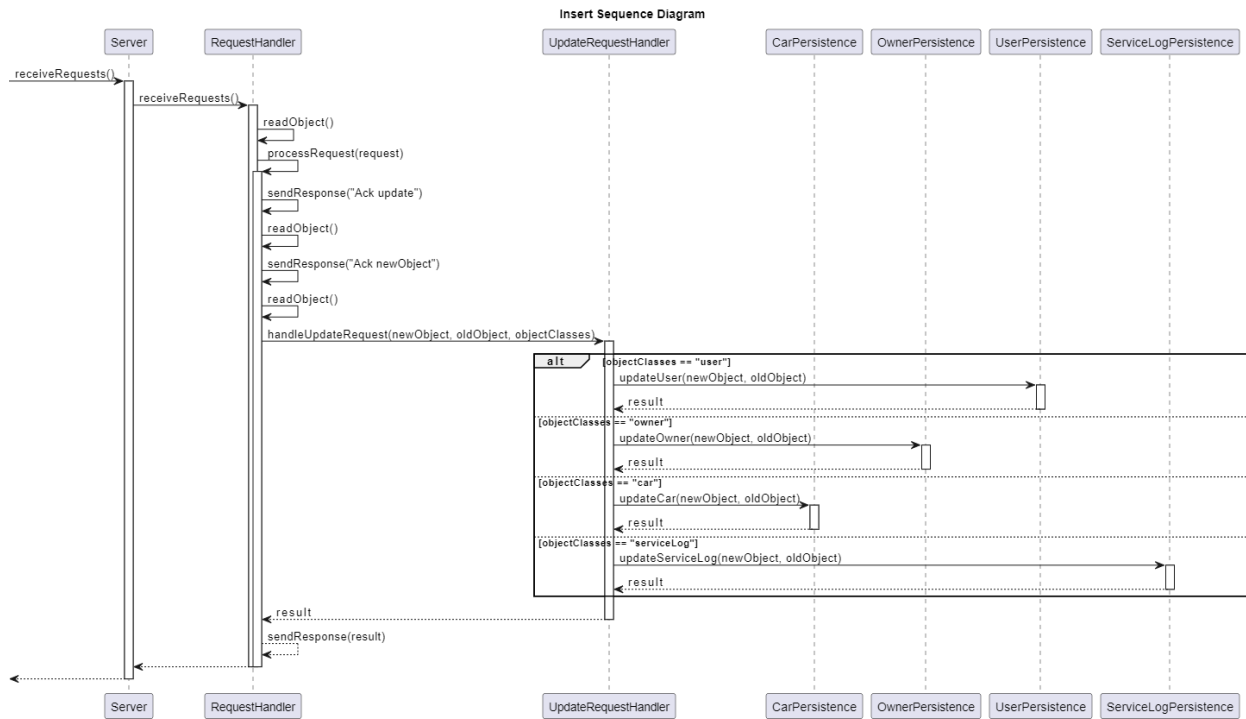


Figure 35

To update an object in its corresponding table, an update request is received from the requester, an ack is sent back to signal that the server is ready to receive the old object to be updated, the object is received, an ack is sent to signal that the server is ready to receive the new object and based on its class, its corresponding persistence object is created to execute the update in the correct table.

### 3.5.2.4. Sequence diagram for deleting a given object from its corresponding table

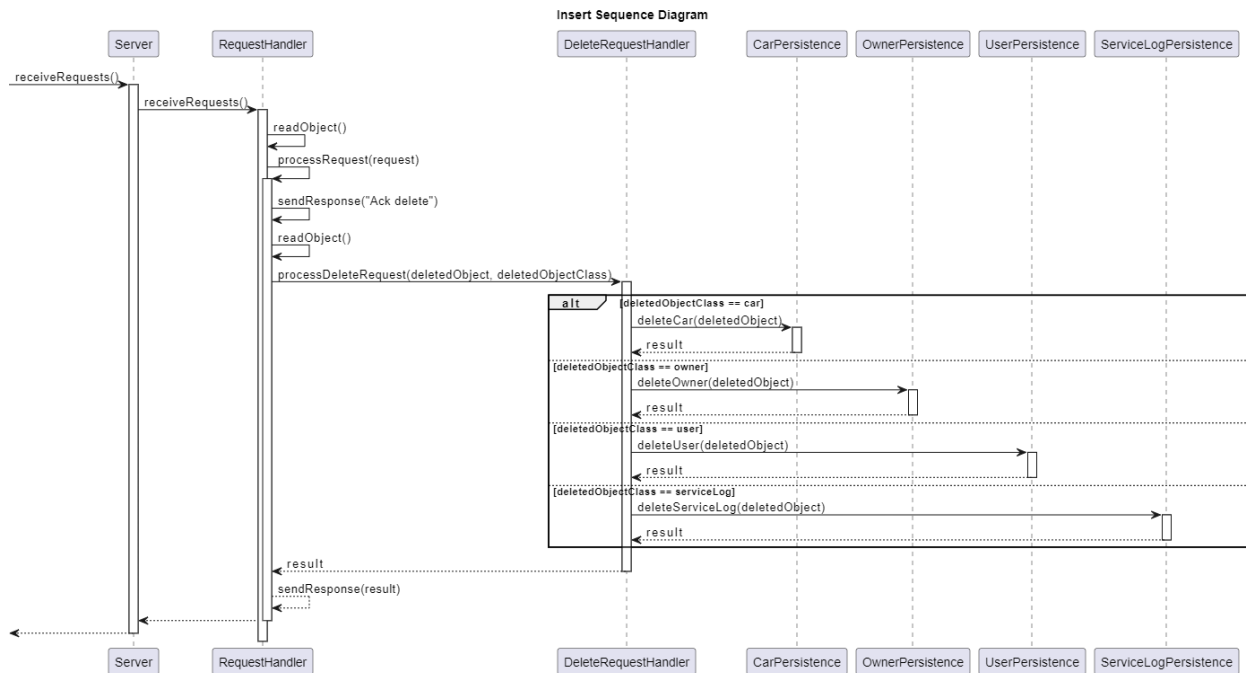


Figure 36

To delete an object from its corresponding table, a delete request is received from the requester, an ack is sent back to signal that the server is ready to receive the object to be deleted, the object is received, based on its class, its corresponding persistence object is created to execute the update in the correct table.

### 3.5.2.5. Sequence diagram for sending a requested list to the requester

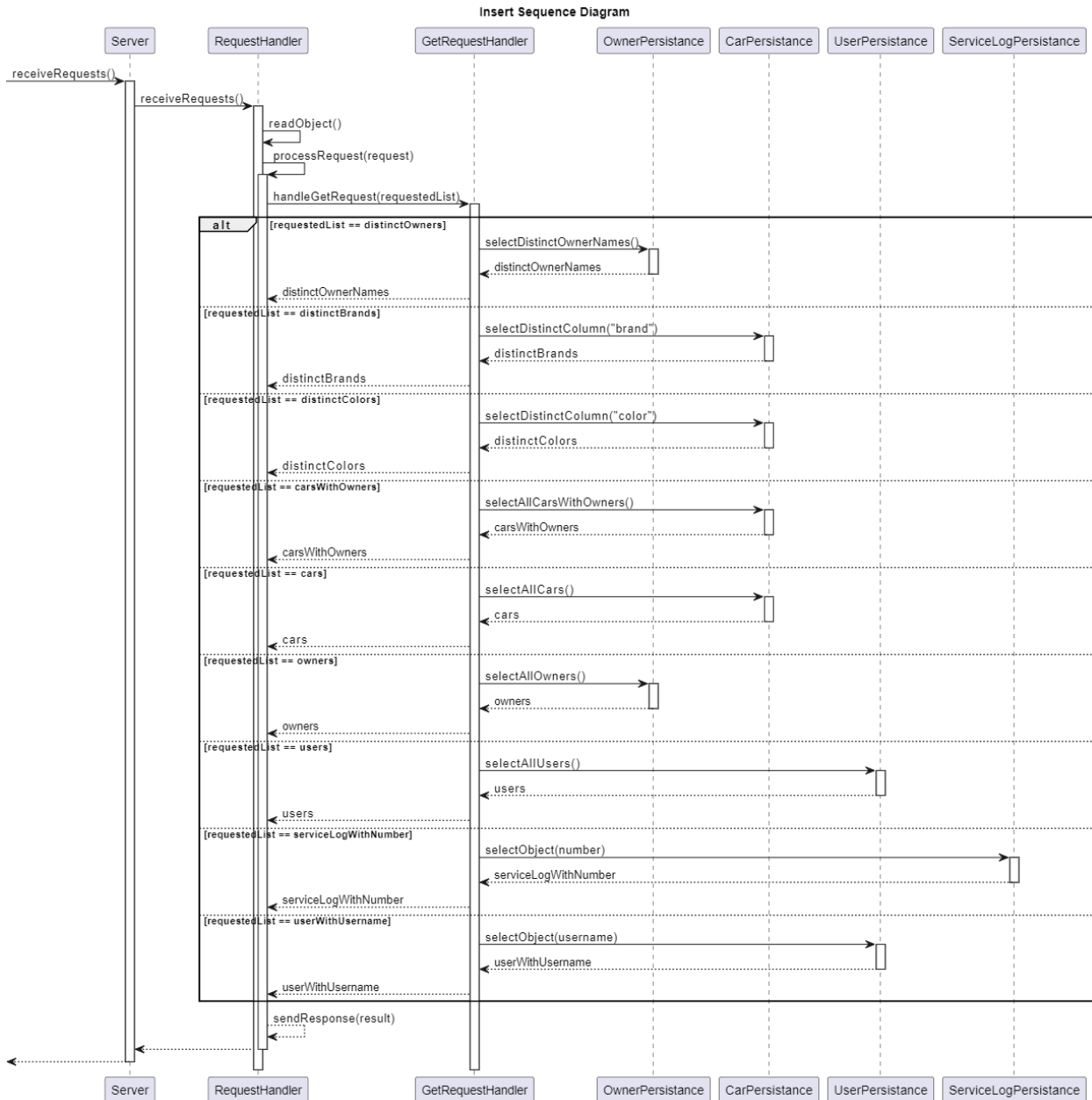


Figure 37

To send a requested list to the requester, the server receives a request with information about which list the requester needs to get, based on the information, the corresponding persistence object is created and all the information is selected and sent back to the requester.



## 4. Design patterns

In the implementation of the application, I used several design patterns to address certain functionalities better. The design patterns I used are the following:

### 4.1. Singleton

I used the singleton design pattern to allow only one instance of the Language object, since the whole application is rendered in a single language at a given time. Whenever a view is rendered, the text is taken dynamically using the selected language. So, when the language is changed and the views are refreshed/re-rendered, they will always be in the correct selected language.

### 4.2. Observer

I used the observer design pattern to notify the active views whenever the language has been changed. Thus, if multiple views are active at a given time and the user changes the language only from one view, the others will also change language to the selected one.

### 4.3. Decorator

I used the decorator design pattern to create the available notification methods towards a user (when its account details have been changed) at runtime. If the email address is not valid, it is unnecessary to try to send an email to an invalid address. The same is true for the phone number.

### 4.4. Façade

I used the façade design pattern to create a “black box” around the notification functionality. Thus, the user controller is not involved in verifying the validity of the user’s email address and phone number.

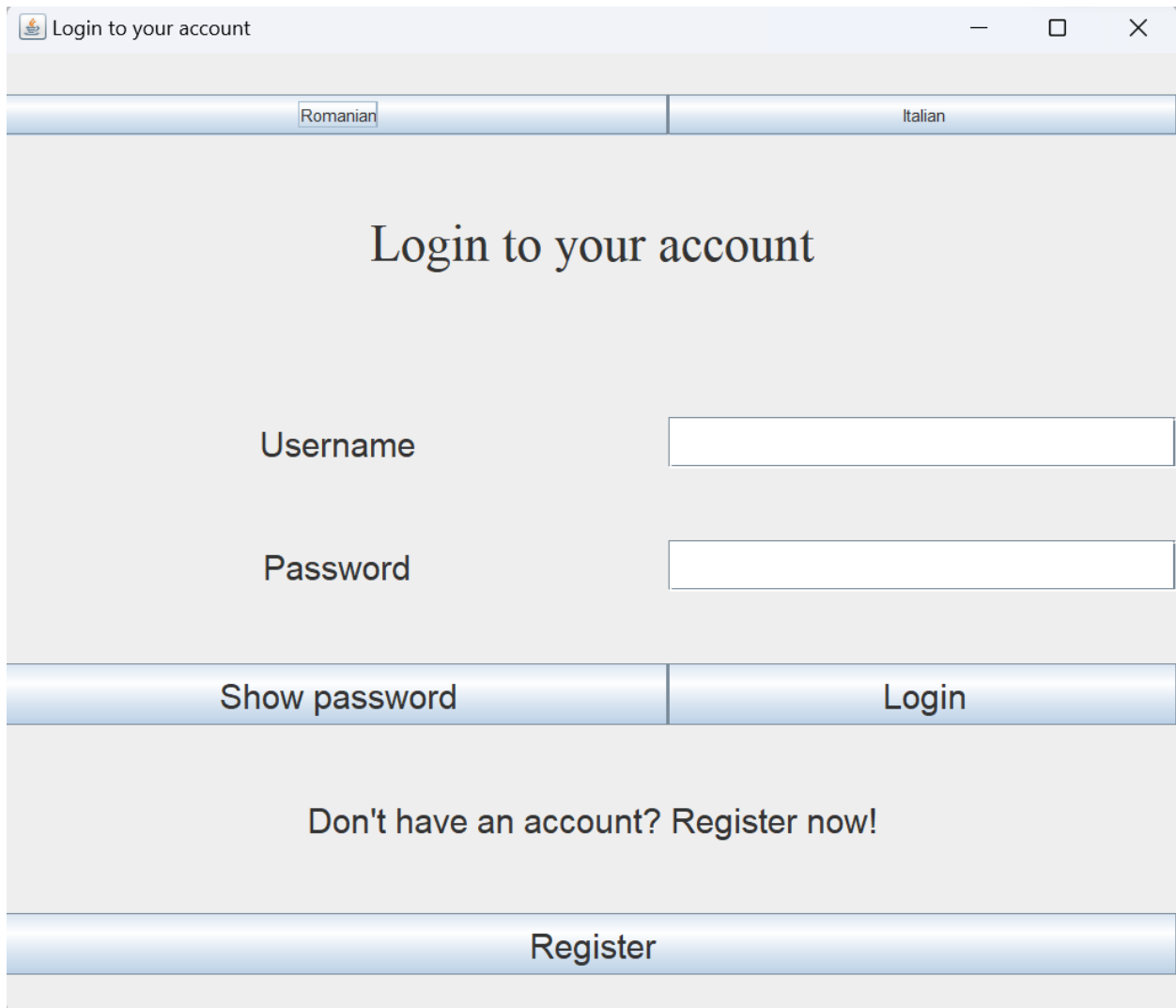
### 4.5. Proxy

I used the proxy design pattern to limit the access to the client instance, thus limiting the access to the server. Also, by using the proxy design pattern, I implemented a cache which stores the received information from the server whenever a request is sent. This improves performance by limiting the amount of requests that are done towards the server.

## 5. Implementation

### 5.1. 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.



Login to your account

Romanian Italian

# Login to your account

Username

Password

Show password Login

Don't have an account? Register now!

Register

Figure 38

## 5.2. 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:

The screenshot shows a window titled "Employee window" with a header bar containing "Romanian" and "Italian" tabs. Below the header, the title "Employee window" is centered. A table displays car information with the following data:

serviceNumber	carId	brand	modelName	color	fuelType	cnp	carOwner
1	1	mercedes	cls	black	diesel	2	suciu cezar
2	2	nissan	silvia s15	white	gasoline	1	name surname
3	3	nissan	gtr	black	diesel	1	name surname
4	4	bmw	e36	purple	gasoline	4	one one

Below the table, there are several controls:

- "Sort car table by:" with checkboxes for "Brand" and "Fuel Type".
- "Select filters to apply on the table above:" with dropdown menus for "Owner", "Brand", "Fuel Type", and "Color".
- "Search cars by owner:" with a dropdown menu for "Owner".
- A row of buttons: "Owner operations", "Car operations", "Service operations", and "Save car info in format".
- A dropdown menu for "csv" next to the "Save car info in format" button.
- A "Log Out" button at the bottom right.

Figure 39

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

Romanian Italian

### Owner operations

cnp	name	surname
1	name	surname
2	suciu	cezar
3	adi	adi
4	one	one

Add a new Owner

Update selected Owner

Delete selected Owner

Figure 40

Car operations

Romanian

Italian

Car operations

id	brand	modelName	color	fuelType
1	mercedes	cls	black	diesel
10	mercedes	g63	black	diesel
2	nissan	silvia s15	white	gasoline
3	nissan	gtr	black	diesel
4	bmw	e36	purple	gasoline
5	bmw	m3	blue	diesel
6	audi	e-tron	yellow	electric
7	mazda	mx5	red	gasoline
8	mazda	rx7	grey	gasoline
9	ferrari	f40	red	gasoline

Add a new Car

Update selected Car

Delete selected Car

Figure 41

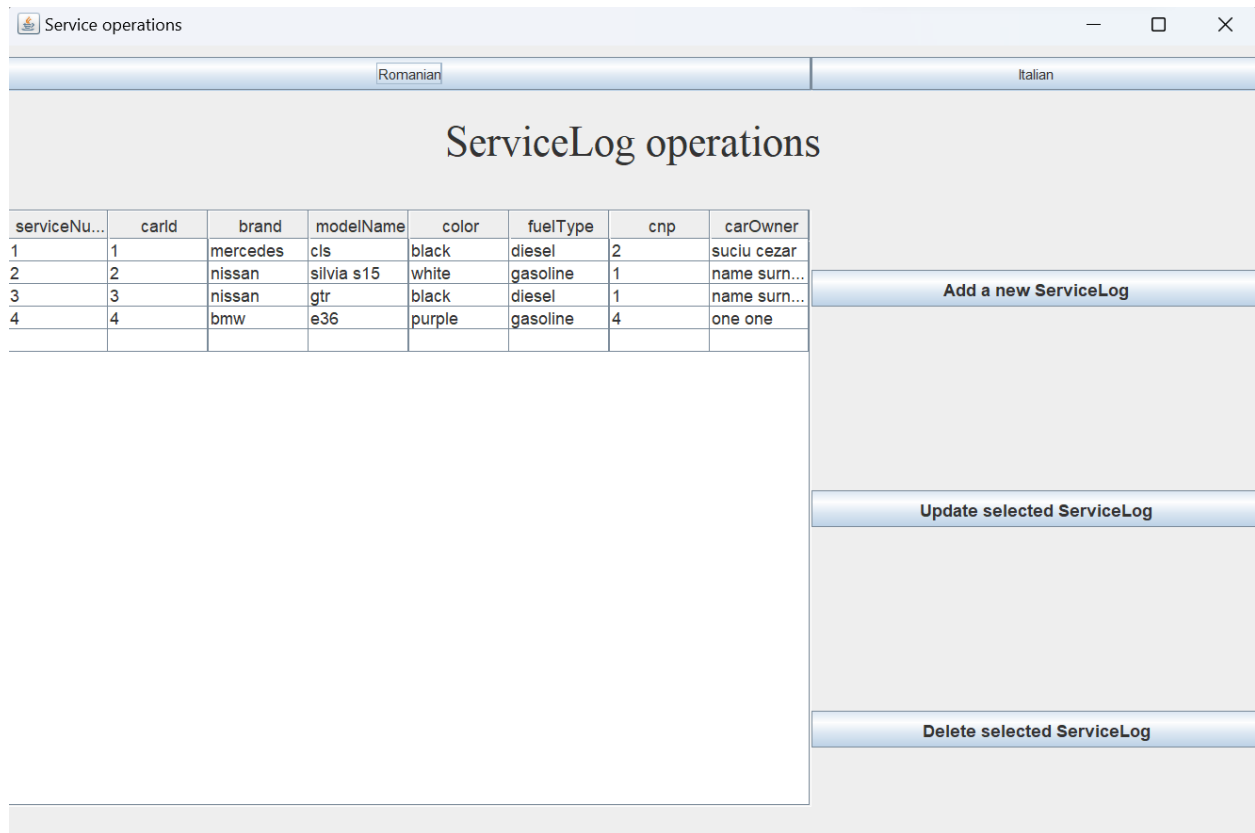


Figure 42

### 5.3. 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 View

Romanian

Italian

## Manager View

serviceNumber	carId	brand	modelName	color	fuelType	cnp	carOwner
1	1	mercedes	cls	black	diesel	2	suciu cezar
2	2	nissan	silvia s15	white	gasoline	1	name surname
3	3	nissan	gtr	black	diesel	1	name surname
4	4	bmw	e36	purple	gasoline	4	one one

Sort car table by:

☐ Brand
 ☐ Fuel Type

Select filters to apply on the table above:

Owner

Brand

Fuel Type

Color

Search cars by owner:

Owner

Save car info in format

csv

Car statistics

Log Out

Figure 43

#### 5.4. 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

RomanianItalian

Admin Window

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

Filter users by:

Role

User operationsLog Out

Figure 44

User operations

RomanianItalian

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

Figure 45