

Frankfurt University of Applied Sciences

FB2: Informatik & Ingenieurwissenschaften

Studiengang Informatik (B.Sc.)



Technical Report

Topic: Car-Selling Platform

Modul: Programming Exercises

Submitted to: Ralf-Oliver Mevius

Summer Semester 2023

Authors:

Ali Al-Haidary (1353109)

Elia Funk (1367315)

Labinot Cekaj (1295092)

Jona Beckmann (1346239)

Submission Date: 29.06.2023

Contents

1. Motivation.....	1
2. Program Introduction	2
2.1 Description	2
2.2 Frontend Implementation	3
2.2.1 User Interface.....	3
2.3 Backend Implementation.....	3
2.3.1 Application Structure and Design.....	3
2.3.2 Application Security	4
3. Architecture and Design.....	5
3.1 ER-Model / Datenbank Schema	6
3.2 DatabaseHandler Class.....	8
3.3 UML-Classdiagram.....	9
3.4 Activity Diagram.....	10
4. Requirement Analysis.....	12
4.1 Requirements.....	12
4.1.1 Login.....	12
4.1.2 Registration	17
4.1.3 Forgot Password.....	22
4.1.4 Dashboard	27
4.1.5 View a specific Listing	34
4.1.6 Create Listings	41
4.1.7 My Listings	47
4.1.8 Edit Profile.....	52
4.1.9 Edit Listings.....	60
4.2 Communication via Mail.....	66
5. Sources	67

List of Figures

Figure 1: Hash-Class.....	4
Figure 2: ER-Model.....	6
Figure 3: DatabaseHandler-Class	8
Figure 4: Complete Class Diagram.....	9
Figure 5: Simple Activity Diagram.....	10
Figure 6: Snowcard-Login	12
Figure 7: User Interface-Login	13
Figure 8: Activity Diagram-Login.....	14
Figure 9: Sequence Diagram-Login.....	15
Figure 10: Snowcard-Registration	17
Figure 11: Use Case Diagram-Registration	18
Figure 12: User Interface-Registration	19
Figure 13: Activity Diagram-Registration	20
Figure 14: Sequence Diagram-Registration.....	21
Figure 15: Snowcard-Forgot Password.....	22
Figure 16: Use Case-Forgot Password.....	23
Figure 17: User Interface-Reset Password.....	24
Figure 18: Reset Password-E-Mail	24
Figure 19: Sequence Diagram-Forgot Password	25
Figure 20: Activity Diagram-Forgot Password.....	26
Figure 21: Snowcard-Dashboard	27
Figure 22: Use Case-Dashboard	28
Figure 23: User Interface-Dashboard.....	29
Figure 24: User Interface-Dashboard with filter.....	30
Figure 25: User Interface-Dashboard no result.....	31
Figure 26: Activity Diagram-Dashboard	32
Figure 27: Sequence Diagram-Dashboard	33
Figure 28: Snowcard-View Listing.....	34
Figure 29: Use Case Diagram-View Listing.....	34
Figure 30: User Interface- Listing View	35
Figure 31: User Interface-Listing View extend Information	35
Figure 32: User Interface-Listing View E-Mail-Text Field.....	36

Figure 33: Listing View - E-Mail send to Seller.....	36
Figure 34: Activity Diagram-Listing View.....	37
Figure 35: Sequence Diagram-Listing View	38
Figure 36: Select Listing and User-Code.....	39
Figure 37: Select Seller Email and Listing-Code	39
Figure 38: Snowcard-Create Listing.....	41
Figure 39: Use Case Diagram-Create Listing.....	42
Figure 40: User Interface-Create Listing	43
Figure 41: Activity Diagram-Create Listing.....	44
Figure 42: Sequence Diagram-Create Listing.....	46
Figure 43: Snowcard-My Listings	47
Figure 44: Use Case Diagram-My Listings	48
Figure 45: User Interface-My Listings	49
Figure 46: Activity Diagram-My Listings	50
Figure 47: Sequence Diagram-My Listings.....	51
Figure 48: Snowcard-Edit Profil.....	52
Figure 49: Use Case-Edit Profile	53
Figure 50: User Interface-Edit Profile	54
Figure 51: User Interface-Edit Profile update successful	55
Figure 52: User Interface-Edit Profile Password Error.....	56
Figure 53: User Interface-Edit Profile delete User	57
Figure 54: Sequence Diagram-Edit Profile.....	58
Figure 55: Snowcard-Edit Listing.....	60
Figure 56: Use Case Diagram-Edit Listing.....	61
Figure 57: User Interface-Edit Listing.....	62
Figure 58: Activity Diagram-Edit Listing.....	63
Figure 59: Sequence Diagram-Edit Listing	64
Figure 60: Mail-Class	66

Statutory Declaration

We herewith declare that we have completed the present report independently, without making use of other than the specified literature and aids. All parts that were taken from published and non-published texts either verbally or in substance are clearly marked as such. This report has not been presented to any examination office in the same form.

Signatures:

A. Ar-H  , Labinot cekaj , J. Beckmann , Funk

Date: 29.06.202

1. Motivation

In today's digital age, where the world has shifted to online markets, businesses must adapt and embrace the power of the modern applications to thrive. And this is why we chose our project topic—a car selling platform that aims to present a modern, easy, fast and user-friendly way of car trading.

Just as car dealerships have witnessed tremendous success by starting to sell their cars online, the automotive market and its ways of trading have started to change and improve drastically. Our car selling platform presents a way of not only expanding the clientele of car dealerships, but of the automotive trading market, a car selling platform that can be used by anyone, dealerships, and private car sellers.

By creating a digital marketplace for cars, we present a way for car dealerships, independent sellers, and buyers to connect effortlessly in a virtual realm. On this platform car trading is not limited to any business hours and geographical constraints anymore. Our platform allows customers to browse their dream vehicles from the comfort of their own homes, at any time that suits them.

Our car selling platform transcends physical boundaries. This application presents an interface to a wider customer base, empowering anyone who wants to sell or buy a vehicle, reach individuals who would have otherwise been out of their reach. This expansion opens new possibilities for growth of businesses and private sellers, into new seemingly unreachable territories.

Working on such a project, did not only enhance our knowledge and programming know-how, but also provided us with invaluable insights into the ways of the new-age online market, as we got inspired by many different available vehicle-selling platforms (like mobile.de). Our team has dugged into the depths of online car markets, understanding what is required to develop and integrating a robust Point of Sale System.

In conclusion, our team has evolved on this journey, which gave us insights on developing a software project as a team. By this experience we received new knowledge in the areas of teamwork, software development and online sales markets and their requirements.

2. Program Introduction

2.1 Description

The Car-Selling Platform is a locally installed software program that uses a cloud-based database to simplify the process of buying and selling cars.

One of the key features of the application is the secure user authentication. It allows users to create a new account, log into the account or reset a forgotten password with ease.

The application provides users with the ability to create and edit a car listing. Users can input technical and general details about the car to create a new listing. They also have the flexibility to edit their existing listings, allowing them to update any information or delete it.

In addition to creating and editing listings, the application offers a comprehensive list of all car listings available in the cloud database. This feature simplifies the process of finding and comparing cars. Furthermore, while viewing a listing, users can send a direct message to the seller, thereby facilitating communication between buyers and sellers and enhancing the overall user experience.

Users also can edit their personal profile, which includes updating their contact information. The system is designed to send automatic email notifications in response to certain user actions, such as creating or deleting an account. This ensures that users are kept informed about important changes to their account.

The dashboard serves as the central hub for all user activities, providing a clear overview of the application's functionalities and allowing for easy navigation between different sections. From the dashboard, users can choose to log out, effectively ending their session. The logout feature is designed to protect user accounts by ensuring that sessions are not left open when not in use.

2.2 Frontend Implementation

The frontend of this project was designed with a primary focus on delivering a user-friendly interface and an intuitive user experience, leveraging the robust capabilities of the JavaFX software platform.

2.2.1 User Interface

JavaFX, a software platform creating and delivering desktop applications, was chosen as the framework for frontend development. The choice of JavaFX facilitated a streamlined approach to building GUI applications, offering a comprehensive suite of tools and libraries to aid development.

A significant part of the user interface design was accomplished using JavaFX Scene Builder software. This visual layout tool simplified the process of creating the layout for our JavaFX application, allowing us to construct the interface via an intuitive drag-and-drop mechanism. This tool significantly enhanced our ability to design and iterate on the user interface, enabling us to experiment with different layouts quickly and efficiently.

2.3 Backend Implementation

In this project, we developed a standalone application, thus the term "backend" primarily refers to the application layer handling business logic, data processing and interactions with local storage or the database.

2.3.1 Application Structure and Design

To structure our application, we employed the Model-View-Controller (MVC) design pattern within the JavaFX architecture. The adoption of this design pattern provided an organized and efficient way to separate concerns within our application, ensuring a clear distinction between application logic (Model), presentation-related code (View), and control logic (Controller).

The Model in our application consisted of simple Java classes, encapsulating the data and state of the application. The View is responsible for rendering the data from the Model in a way that is suitable for interaction. It defines exactly how the application's data should be presented and is responsible for updating the presentation when the data changes. For our application, the View was primarily constructed using FXML, a declarative markup language specific to

JavaFX. FXML allowed us to define the user interface layout in a way that was easy to understand, manage, and modify, making it an excellent choice for creating user interfaces.

The Controller served as the intermediary between the Model and the View. It was responsible for processing user interactions, updating the Model accordingly, and reflecting Model changes in the View.

This architectural approach greatly simplified the development process, made the code easier to understand and maintain, and provided a clear roadmap for potential future enhancements.

2.3.2 Application Security

2.3.2.1 Hash

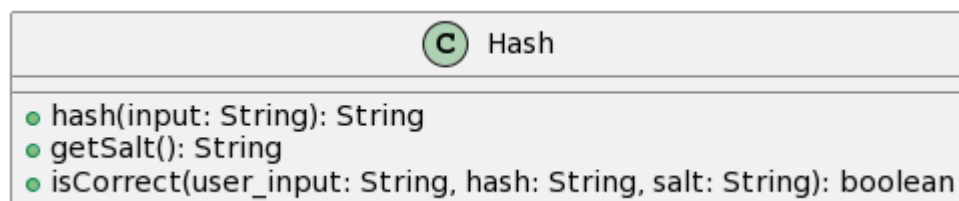


Figure 1: Hash-Class

The Hash class is used by us to guarantee the security of the data of our customers. At the core of the class are three methods, each performing a unique function. The first, the hash method, employs the SHA-256 algorithm to transform a string into a unique hash value, presented as a hexadecimal string. This input could be a password or any other piece of text.

Following the hash method is the `getSalt()`, which randomly generates a 32-character string known as a salt. Made up of alphanumeric characters and special symbols, these salts serve to add an extra layer of complexity to the hashing process, thereby enhancing the security of passwords.

The `isCorrect()` has the critical role of validating whether a user's input, combined with a given salt value, aligns with a pre-stored hashed password and salt. The method performs this check by hashing the user's input together with the specified salt and then comparing this outcome with the original hash value. If the result matches, it returns true, indicating correctness. Otherwise, it will return false.

The `hash()` function is employed in conjunction with the `getSalt()` function to first concatenate the password with a randomly produced salt and hash it. Doing so ensures that passwords aren't

stored in their plain text form in the database, but rather as hashes. Utilizing salts offers an additional layer of security, safeguarding these hashes from vulnerabilities such as rainbow table attacks.

Additionally, the `getSalt()` function plays a vital role in generating a one-time code for password resetting. However, in this context, only the final 6 characters of the salt are utilized to create the one-time code.

2.3.2.2 Config File

The `Config` class in Java provides a robust and simple mechanism to securely store sensitive information like passwords, and it keeps them local instead of uploading to GitHub. By following a key-value pair structure, the class dynamically retrieves specific properties from the loaded configuration.

One of the key functionalities of this class is the `getEmailUsername()` method, which fetches the value of the property having the key "email.username". This means you can easily retrieve the username for an email, stored securely in your local environment. Similarly, the `getEmailPassword()` method returns the password associated with the email, based on the "email.password" key in the properties.

The class also caters to database connections. `getDatabaseUrl()`, `getDatabaseUser()`, and `getDatabasePassword()` are methods that return the values of their corresponding properties, namely "database.url", "database.user", and "database.password". These methods make it easy to retrieve and utilize necessary database credentials without exposing them in the code.

By using this class, we can store and isolate sensitive information, such as email credentials and database connection details, in the `config.properties` file and access them securely within the application. This approach allows for separation of sensitive data from the source code, enhancing security and facilitating configuration changes without modifying the code itself.

3. Architecture and Design

3.1 ER-Model / Datenbank Schema

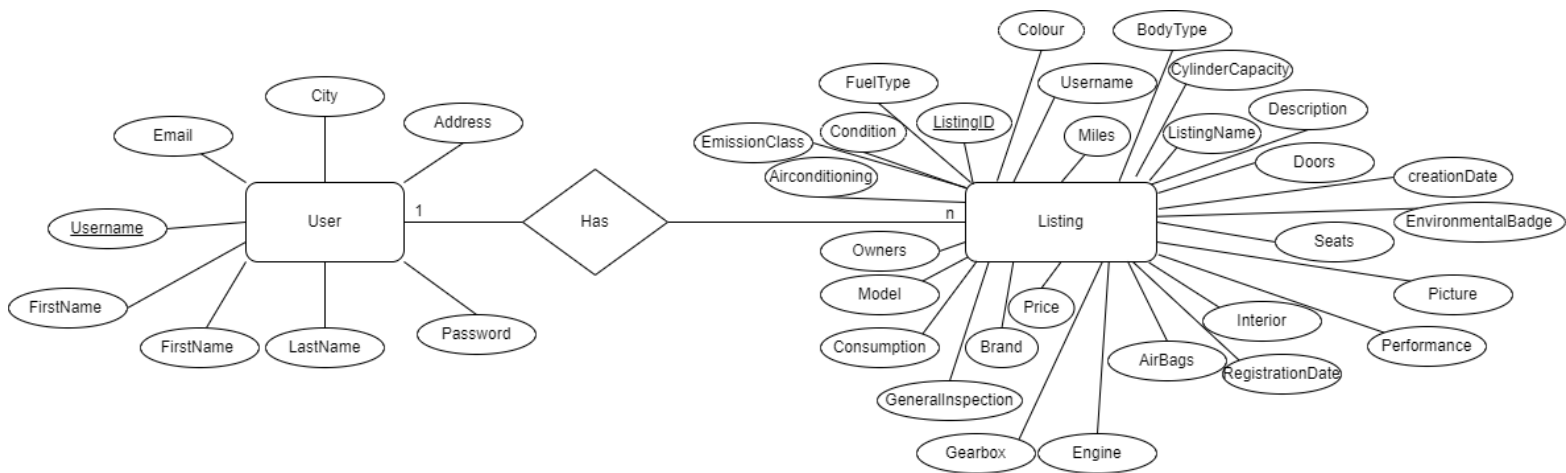


Figure 2: ER-Model

Our database consists of two tables, "Listing" and "User," which are related in a one-to-many (1:N) relationship. The "Listing" table represents individual listings, while the "User" table stores user information.

Table: User

- Primary Key: Username (Unique identifier for each user)
- Other Columns of Note: FirstName, LastName, Email, City, Address, Password, Salt

Table: Listing

- Primary Key: ListingID (Unique identifier for each listing)
- Foreign Key: Username (References the Username column in the User table)
- Other Columns of Note: ListingName, creationDate, Price, Brand, Model

The "User" table stores user details, including FirstName, LastName, Email, City, Address, Password, and Salt. The Username column serves as the primary key, guaranteeing a unique identifier for each user.

The "Listing" table contains essential information about each listing, such as the ListingID, ListingName, creationDate, Price, Brand, and Model. The ListingID serves as the primary key, ensuring a unique identifier for each listing. The Username column acts as a foreign key, establishing a relationship with the Username column in the User table. This key ensures that each listing is associated with a valid user.

By utilizing the primary key and foreign key constraints, we establish a connection between the "Listing" and "User" tables. This relationship enables us to link listings to their respective users and retrieve information from both tables when needed.

In addition to the table descriptions, I would like to mention that we have implemented a Java class called "DatabaseHandler" to manipulate the database using the JDBC API. The DatabaseHandler class follows the Singleton pattern to ensure efficient resource utilization.

The DatabaseHandler class is responsible for establishing and managing the connection to the database, executing SQL queries, and handling transactions. By utilizing the JDBC API, we can interact with the database by creating statements, executing queries, and retrieving results.

The Singleton pattern ensures that only one instance of the DatabaseHandler class is created throughout the application's lifecycle. This approach promotes resource efficiency by reusing the existing instance instead of creating multiple instances, which can be costly in terms of memory and processing power.

Overall, the combination of the JDBC API and the DatabaseHandler class implemented using the Singleton pattern allows for effective and optimized interaction with the database within our Java application.

3.2 DatabaseHandler Class

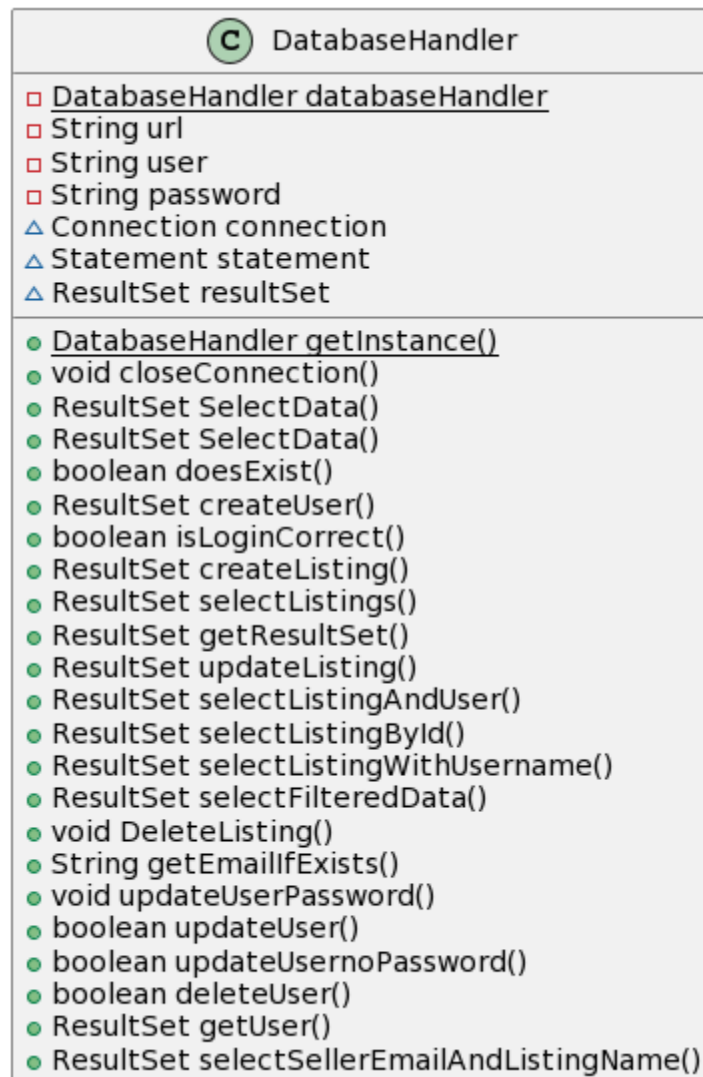


Figure 3: DatabaseHandler-Class

The DatabaseHandler is used to let our program communicate with our database. Because our database provider CleverCloud doesn't allow many connections at the same time, we decided to use the singleton design pattern.

Here's a breakdown of its key components:

- The class maintains a single instance of **DatabaseHandler** using the singleton design pattern.
- The class has instance variables for storing the database URL, username, and password obtained from the **Config** class.

- The **DatabaseHandler** class establishes a connection to the database in its constructor using the URL, username, and password.
- The class provides **multiple methods** for performing various database operations such as selecting data, creating users and listings, updating listings, and fetching specific data from the database.
- The class uses **java.sql** package for handling database connections, executing SQL statements, and retrieving result sets.
- The class also utilizes the **Config** class to retrieve configuration details such as the database URL, username, and password.
- Some of the methods in the class use prepared statements to execute parameterized SQL queries.

3.3 UML-Classdiagram

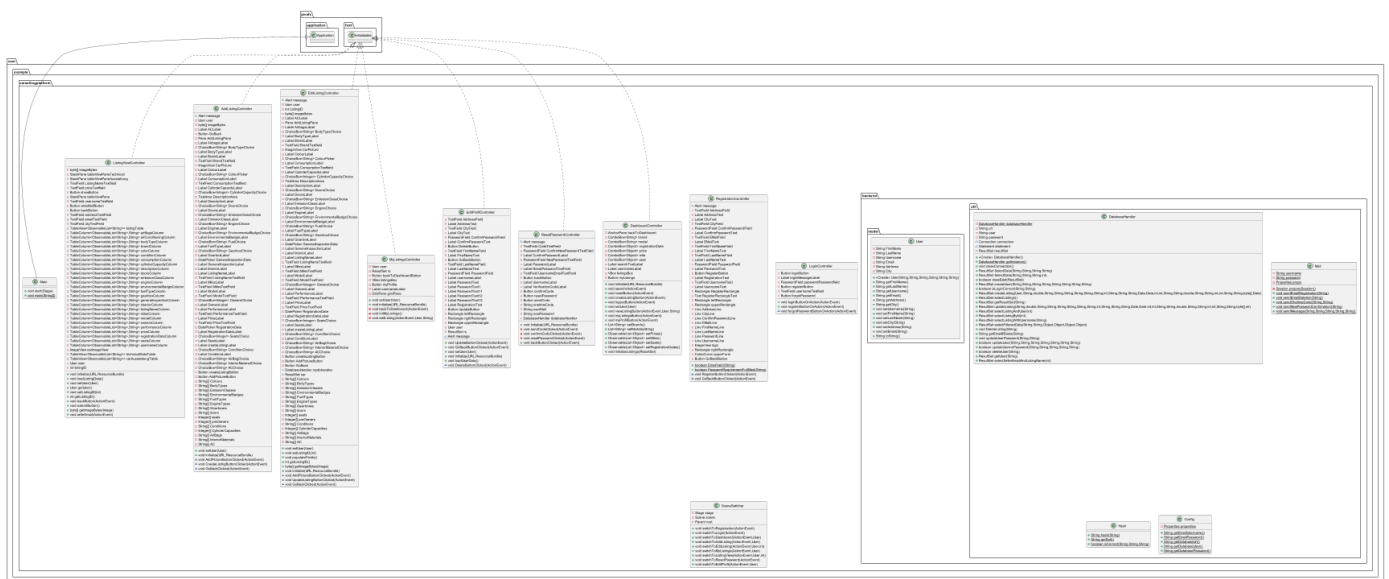


Figure 4: Complete Class Diagram

This diagram gives a complete overview of the application. All these classes play a significant role in fulfilling several requirements this project brought with itself. A more detailed view on how these classes are used in this project, will be provided in the following documentation, in form of requirement analysis.

3.4 Activity Diagram

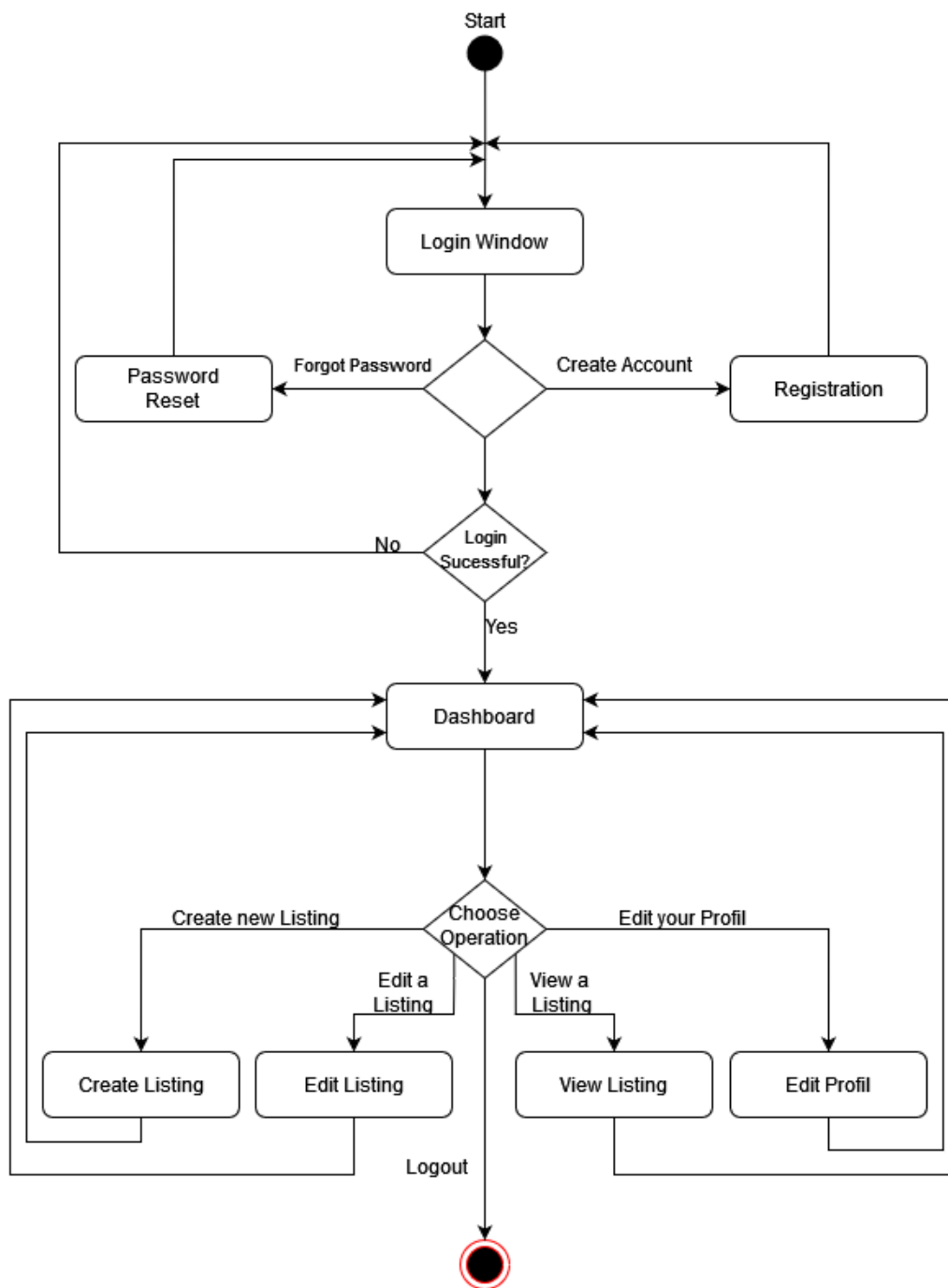


Figure 5: Simple Activity Diagram

The activity diagram under discussion provides a streamlined depiction of the application's workflow. Upon initiating the program, the user encounters a login interface. Here, the user has the option to either establish a new account, log into the system or in the event of a forgotten password, initiate a password reset by selecting the appropriate button.

Post successful authentication, the user is granted access to a range of functionalities. These include the ability to generate a new car listing, modify an existing car listing, going through the list of car listings and reviewing a listing, or alter their personal profile. Upon completion of the chosen task, the user can return to the dashboard.

The dashboard serves as a central hub, from where the user can choose to log out, thereby concluding the application session. This activity diagram effectively encapsulates the user journey within the application, from login to logout, highlighting the key functionalities and user interactions.

4. Requirement Analysis

4.1 Requirements

4.1.1 Login

4.1.1.1 Snowcard Login

Name	Login
ID	1
Description	The login function is designed to authenticate registered users. To log into the system the user can either use the email address or the username with the corresponding password.
Trigger	The user click on the “Login” button on the Login Frame
Actors	User
Pre-Condition	Connection to the internet, user must know their credentials
Post-Condition	User has successfully logged in and will be redirected to the dashboard frame
Basic flow Description Actions 1 2 3 4	User successfully authenticates using Username or Email User enters Username or Email User enters corresponding password Login Controller and databaseHandler receive the data and compare to the entries in the database Correct credentials and redirected to the dashboard frame
Alternative Flow Description Actions 1 2 3 4 5	A Are the Username and Password fields filled in? User enters Username, Email or Password Login Controller checks if text field and password field are filled in User didn’t enter both fields Error Message displayed User can reenter the credentials
Alternative Flow Description Actions 1 2 3 4 5 6	B Username or Email wasn’t found User enters Username or Email User enters corresponding password Login Controller and databaseHandler receive the data and compare to the entries in the database No fitting Username or Email found in the database Error Message displayed User can reenter the credentials

Figure 6: Snowcard-Login

4.1.1.2 User Interface

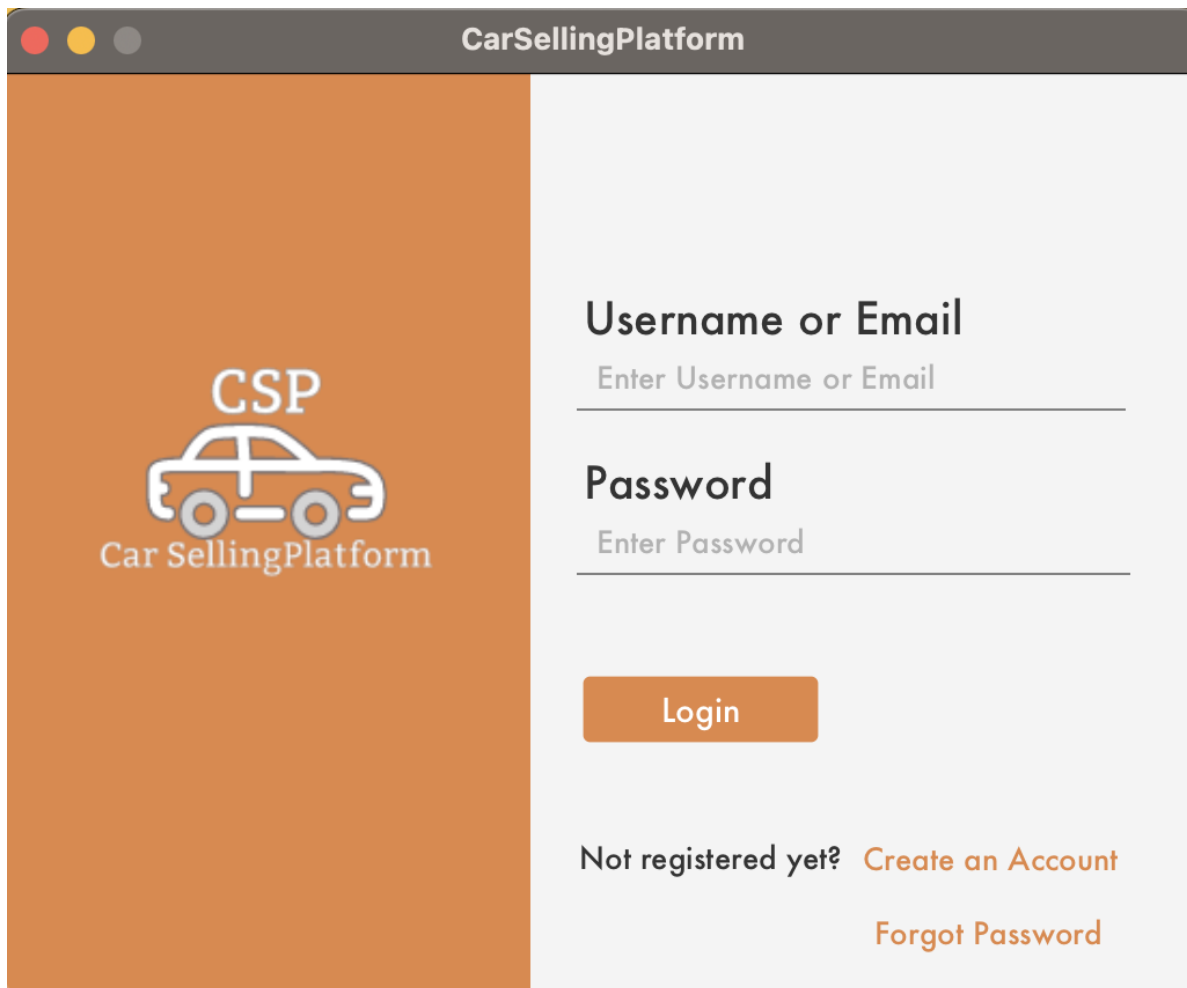
The image shows a login window for a platform called 'CarSellingPlatform'. The window has a dark grey title bar with three colored window control buttons (red, yellow, grey) on the left and the text 'CarSellingPlatform' in the center. The main area is split into two vertical panels. The left panel has an orange background and features a white logo consisting of a car silhouette with 'CSP' above it and 'Car SellingPlatform' below it. The right panel has a light grey background and contains the login form. The form has two input fields: 'Username or Email' with the placeholder text 'Enter Username or Email' and 'Password' with the placeholder text 'Enter Password'. Below these fields is an orange 'Login' button. At the bottom of the right panel, there is a link 'Not registered yet? Create an Account' and another link 'Forgot Password'.

Figure 7: User Interface-Login

The login window is designed to provide users with an efficient way to access their account. This interface comprises different elements that facilitate login, registration and password recovery processes.

To log in, the user must input either their username or email address along with the corresponding password. After entering the login details, the user can click on the “Login” button to access their account. If the user does not have an account, they can create a new one by clicking on the “Create an Account” button. This button redirects the user to a separate registration page. This is also the case when clicking on the “Forgot password” button.

4.1.1.3 Activity Diagram

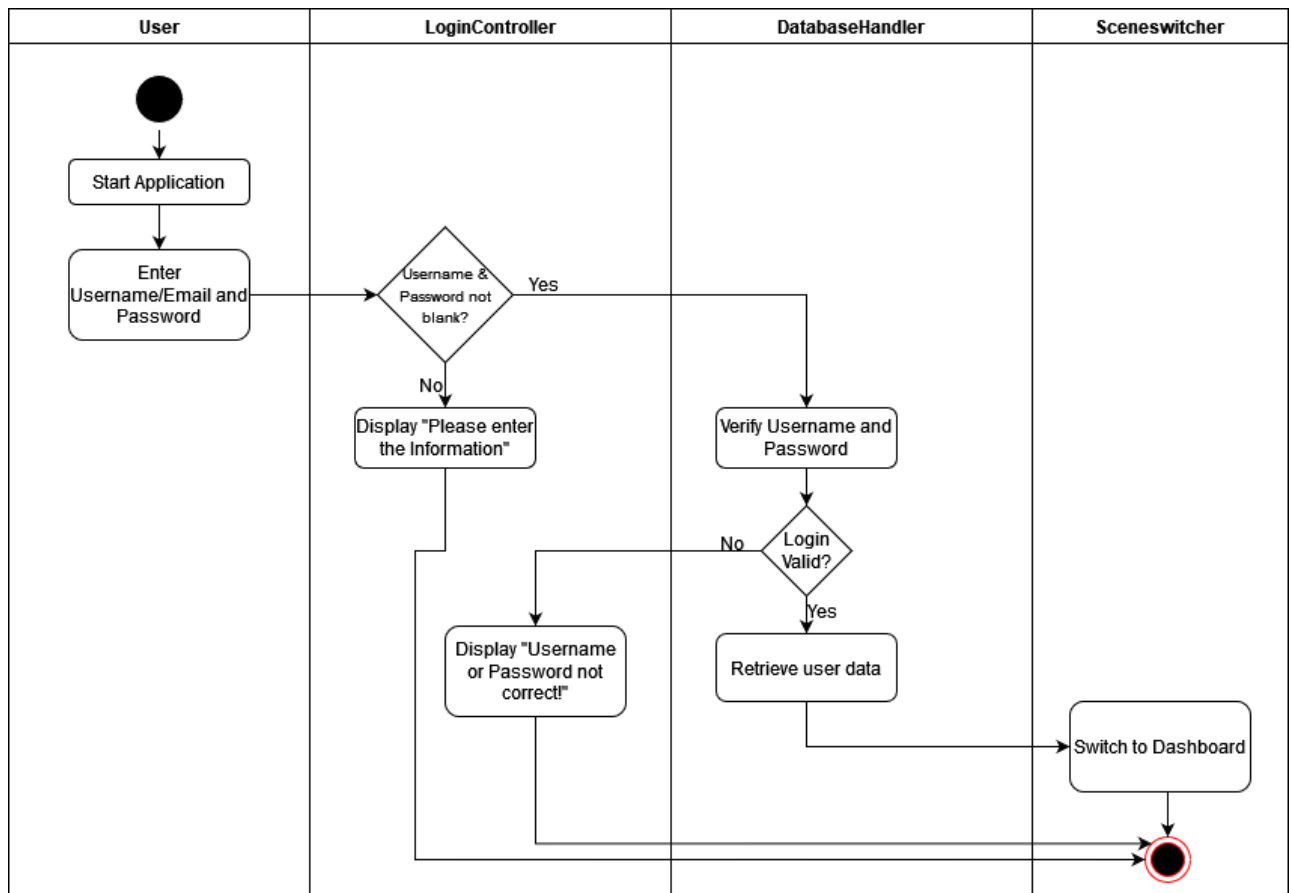


Figure 8: Activity Diagram-Login

The activity diagram provides a comprehensive view of the login process in the car selling platform, detailing the flow of operations within the `loginButtonOnAction` method in the `LoginController` class.

After starting the application, the user enters their username or email with the password. After clicking the login button, the system checks if the two fields are not blank. If either field is blank, the system displays a message prompting the user to enter the required information.

If both fields are filled, the system proceeds to verify the entered username, email and password with the `DatabaseHandler`. This verification involves checking if the entered credentials match the ones stored in the database. If the credentials do not match, the system displays a message indicating that the username or password is incorrect.

If the credentials match, the system retrieves the user data from the database and switches to the dashboard, effectively logging the user into the platform.

4.1.1.4 Sequence Diagram

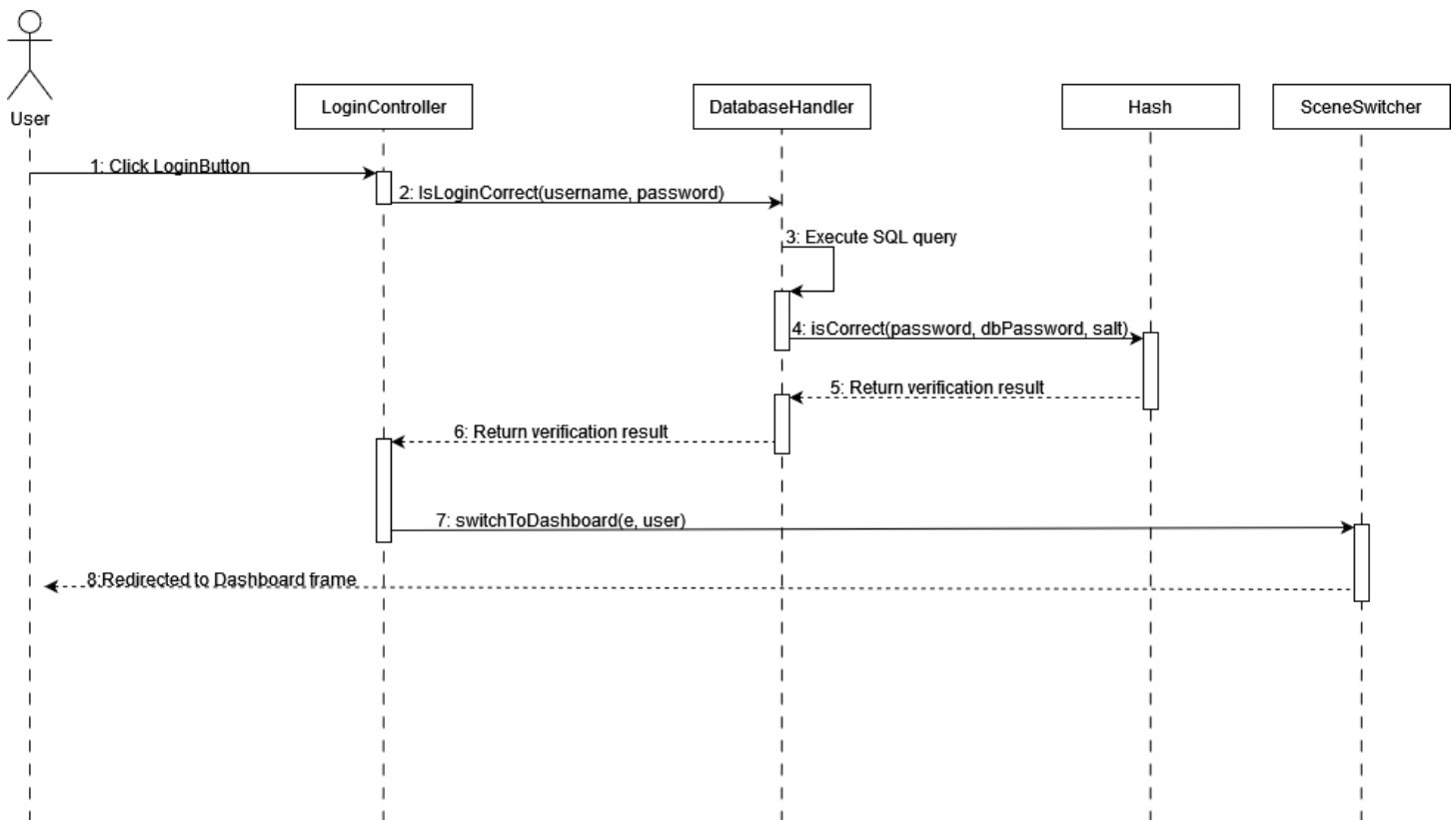


Figure 9: Sequence Diagram-Login

The sequence diagram provides a detailed overview of the login process in the car selling platform, outlining the interactions between the user, the LoginController, the DatabaseHandler, and the Hash class.

The process begins when the user inputs their username and password. This action triggers the `loginButtonOnAction(username, password)` function in the LoginController class. The LoginController then communicates with the DatabaseHandler by calling the `isLoginCorrect(username, password)` function. This function is responsible for verifying the user's credentials.

Inside the DatabaseHandler, an SQL query is executed to retrieve the salt and hashed password associated with the provided username or email from the database. The DatabaseHandler then proceeds to verify the provided password against the hashed password retrieved from the database. This verification is done by calling the `isCorrect(password, dbPassword, salt)` function in the Hash class, passing in the user-provided password, the hashed password from the database, and the associated salt.

The Hash class performs the password verification by hashing the user-provided password with the salt and comparing the result with the hashed password retrieved from the database. The result of this comparison, a boolean indicating whether the passwords match, is then returned to the DatabaseHandler.

The DatabaseHandler subsequently returns the verification result to the LoginController. Depending on the result, the LoginController displays an appropriate message to the user. If the verification was successful, the user is logged in. If not, an error message is displayed.

4.1.2 Registration

4.1.2.1 Snowcard

Name	Registration
ID	2
Description	To be able to use this application in the first place, the users are required to use accounts for this platform. To use these accounts, they must be created first. This requirement assures that the users will be able to create accounts, that grant them the access to the application.
Trigger	User click on the “Create Account” button on the Login Frame
Actors	User
Pre-Condition	Application installed, connection to the application database, user does not own an account linked to his email address
Post-Condition	User has successfully registered and is now owner of an account
Basic flow	
Description	User successfully creates an account
Actions	
1	User clicks on the “Create Account” button on the Login frame
2	The Login opens the Registration window
3	The user enters all of the required data to create an account (First name, last name, password, city, email, username)
4	The Registration Controller and the DatabaseHandler receive the data and check it for mistakes
5	The DatabaseHandler saves the data and creates a new account
6	Registration uses the Mail class and sends confirmation mail to the users entered email address
Alternative Flow	A
Description	User enters an invalid email address
Actions	
1	The user enters an already in use or invalid email address
2	Registration displays a warning message, that informs the user, that his email is invalid
3	User has to re-enter his email address until no warning is displayed
Alternative Flow	B
Description	User enters an email that is already linked to an existing account
Actions	
1	The user enters an email address that is already linked to an existing account
2	Registration Controller displays a warning message, that informs the user, email already in use
3	User enters a different email address or leaves the registration window
Alternative Flow	C
Description	User enters an username that is already linked to an existing account
Actions	
1	The user enters an username that is already linked to an existing account
2	Registration Controller displays a warning message, that informs the user, username already in use
3	User enters a different username or leaves the registration window
Alternative Flow	D
Description	The user wants to use a invalid password format
Actions	
1	The user enters a password that does not meet the applications password requirements
2	Registration displays a warning message, that informs the user which requirements are not fulfilled
3	User chooses a different password that meets all requirements
Alternative Flow	E
Description	Password and password confirmation are not the same
Actions	
1	User enters a different input in the password and confirm password password field
2	Registration Controller displays a warning, which states, that the inputs are not matching
3	The user corrects the mistake and enters the same password twice

Figure 10: Snowcard-Registration

4.1.2.2 Use Case Diagram

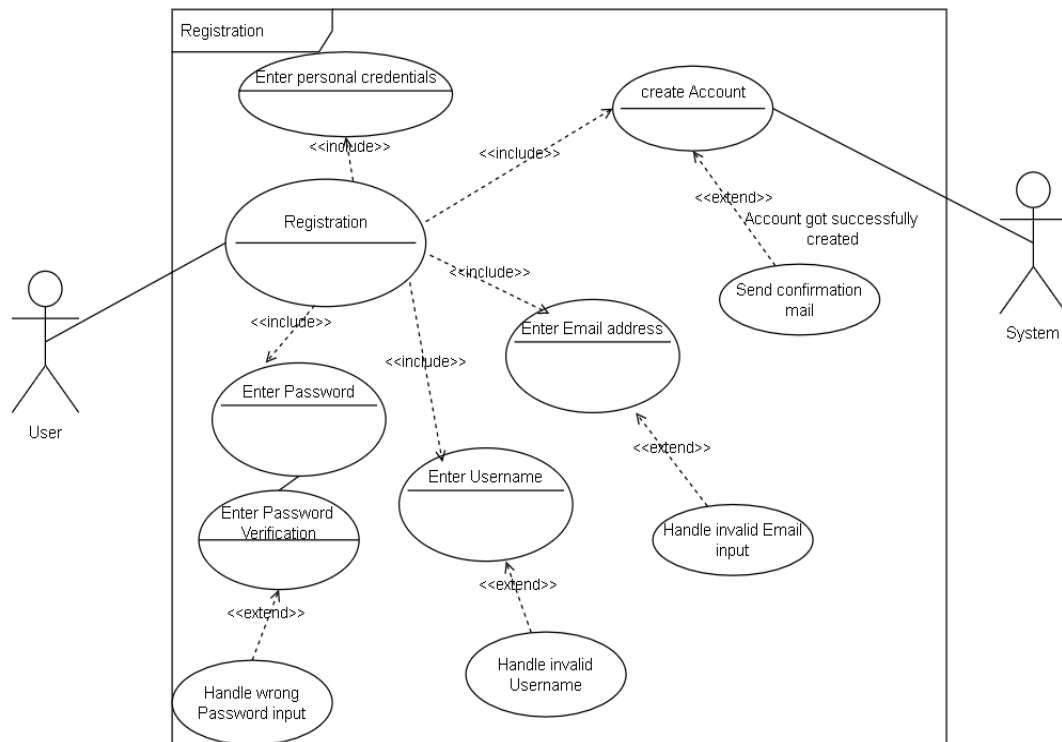


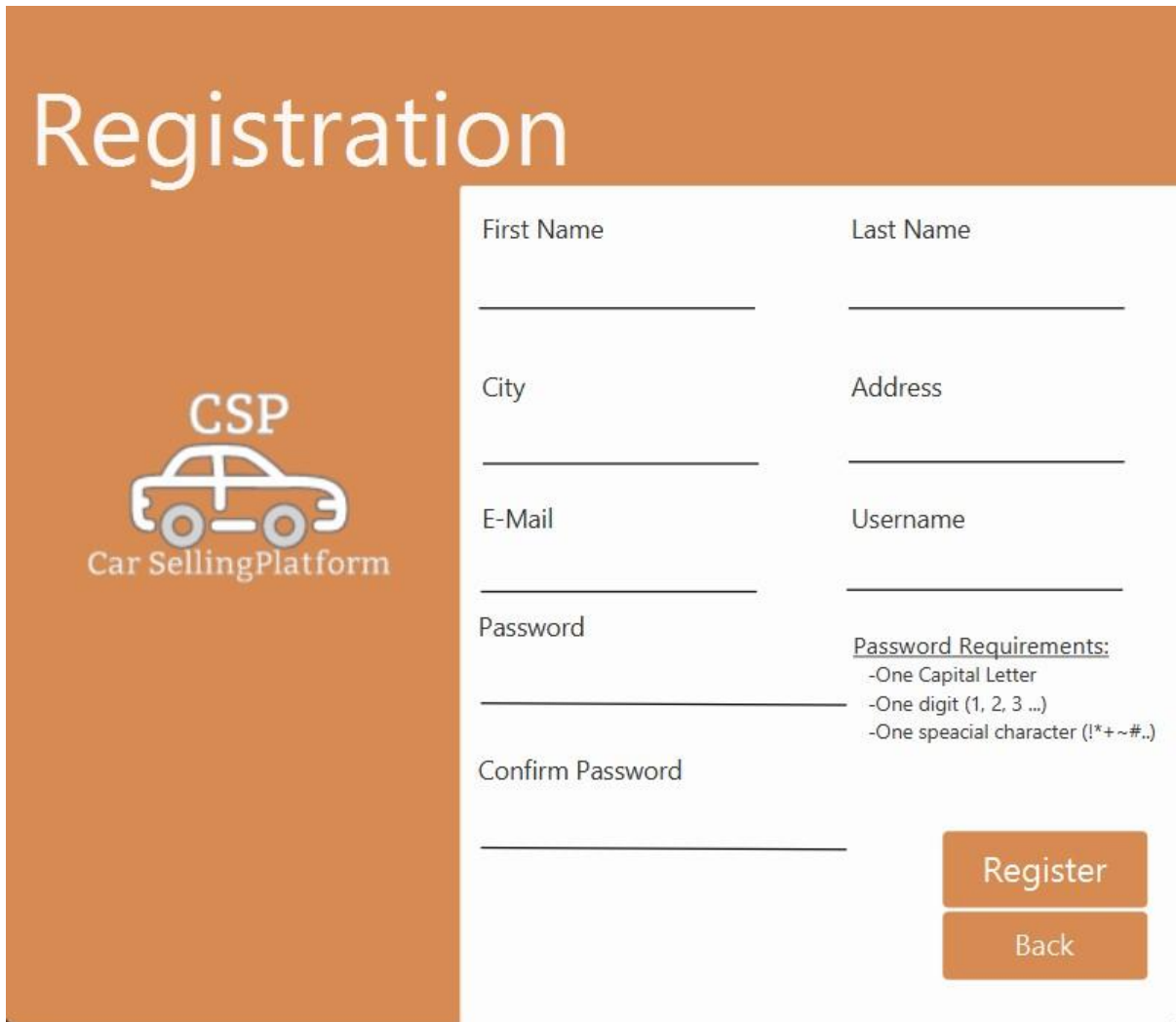
Figure 11: Use Case Diagram-Registration

This use case diagram illustrates how the registration on our platform is supposed to work. As soon as the user opens the registration window, he must enter his personal information (such as first name, last name and address), email address and password (twice for verification).

Any wrong kind of input by the user, like entering an invalid email address, or trying to use an already existing username, will be handled by the Registration class.

If the user is done entering his user data, an account gets created, and a confirmation mail sent to the user's email address.

4.1.2.3 User Interface



The image shows a registration form for the CSP Car Selling Platform. The form is set against an orange background. On the left, there is a logo for CSP Car Selling Platform featuring a stylized car icon. The word 'Registration' is written in large white letters at the top left. The form fields are arranged in two columns. The first column contains fields for First Name, City, E-Mail, Password, and Confirm Password. The second column contains fields for Last Name, Address, Username, and Password Requirements. The Password Requirements are listed as: -One Capital Letter, -One digit (1, 2, 3 ...), and -One special character (!*+~#..). At the bottom right, there are two orange buttons: 'Register' and 'Back'.

Registration

CSP
Car Selling Platform

First Name	Last Name
<input type="text"/>	<input type="text"/>
City	Address
<input type="text"/>	<input type="text"/>
E-Mail	Username
<input type="text"/>	<input type="text"/>
Password	<u>Password Requirements:</u> -One Capital Letter -One digit (1, 2, 3 ...) -One special character (!*+~#..)
<input type="text"/>	
Confirm Password	
<input type="text"/>	

Register

Back

Figure 12: User Interface-Registration

In this picture the Graphical User Interface of this application's Registration can be seen. On the right side, you can see the text fields which are used to get and process the user's input and create an account with it. Right next to the Password Fields you can see a label, which contains all requirements for this application's passwords, and right next to it are two buttons, of which the upper one fulfills this User Interface's actual task (taking user data and creating an account with it), and the button below, gives the user the opportunity to switch back to the login frame, in case he changed his mind and does not want to create an new account anymore.

4.1.2.4 Activity Diagram

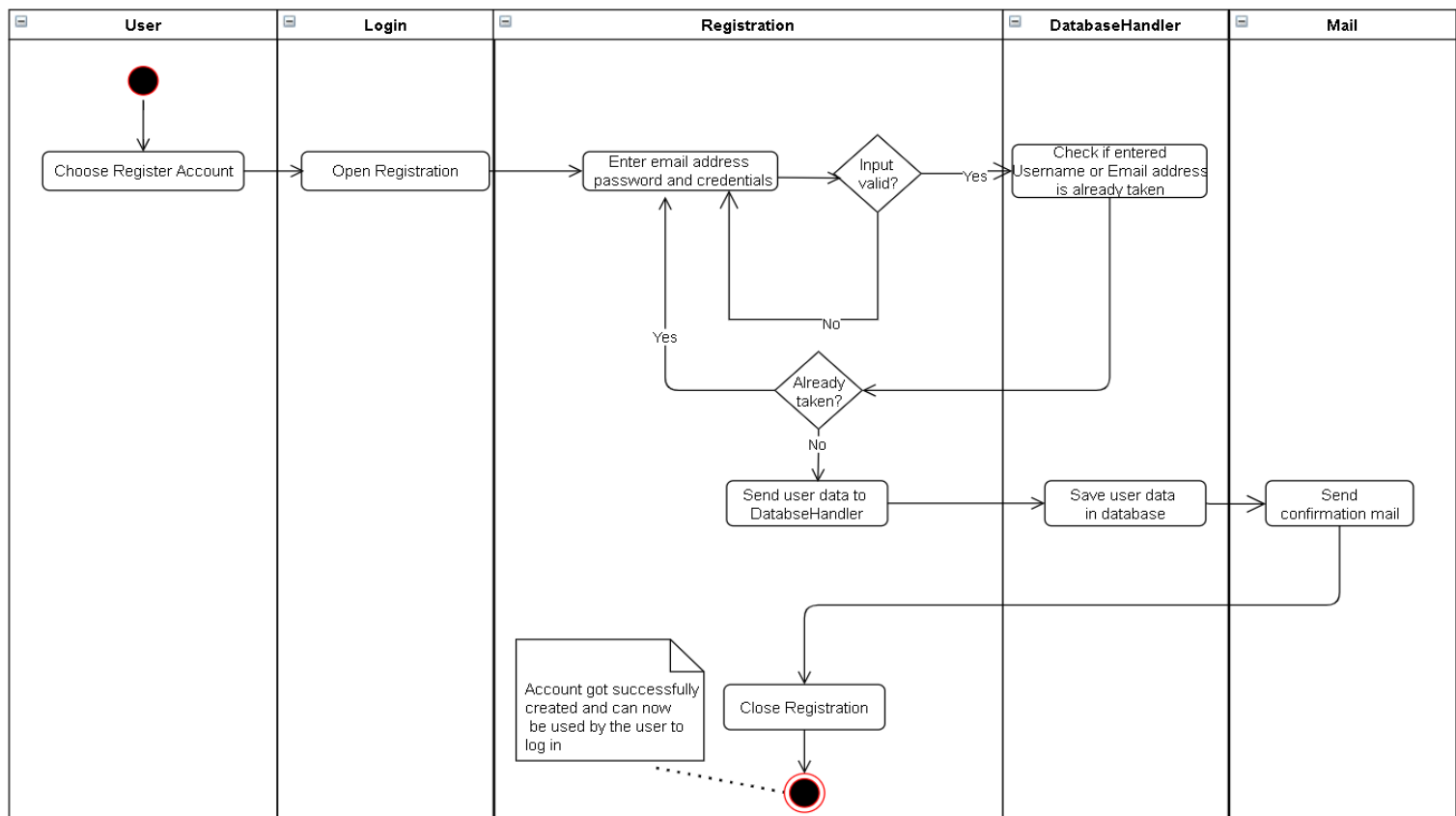


Figure 13: Activity Diagram-Registration

This activity diagram shows the regular registration process for all users of this application. At the beginning, the user gets presented the login frame, from which he can switch to the registration frame by using the “Create Account” button on the login frame. When this happens, the user gets presented the registration frame, on which he must enter his user data, such as his first name, last name, address, his email address, and a username and password of his choice. After entering his data, the RegistrationController checks if the given input is valid. This means that this class first checks if the entered email address is valid, and if the password input matches the password confirmation input, if that is the case, the RegistrationController checks the Database using the DatabaseHandler class, and checks if the username and email address the user wants to use are already linked to an existing account, if any of the users inputs are non-valid, the user gets an error message, and gets another chance to correct himself, if that is not the case, the user's data gets used to create an account which then gets saved on the database. At the end, a confirmation mail will be sent to the newly created account's email address, and the application leads the user back to the login screen, from where he can now log in with his new account.

4.1.2.5 Sequence Diagram

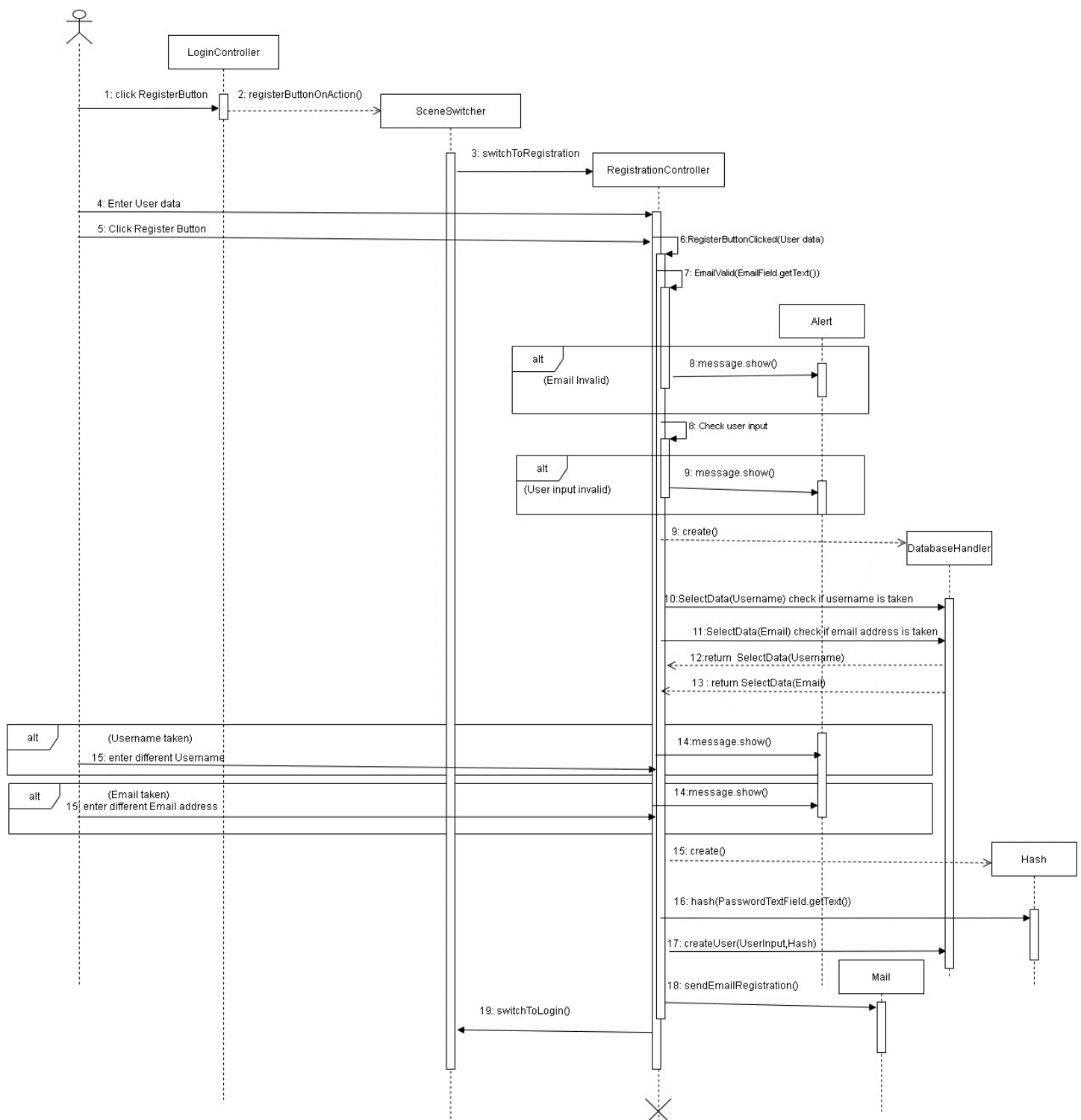


Figure 14: Sequence Diagram-Registration

Here you can see a sequence diagram, which visualizes the normal registration process on this application in a more detailed manner, than the activity diagram above.

Here you can see two extra classes, of which one is the ScenceSwitcher which makes the application switch the scenes/frames, and the hash class, which encrypts the password in SHA-256 with a salt, which leads to an encrypted cipher code being saved on the database instead of a raw password text.

4.1.3 Forgot Password

4.1.3.1 Snowcard

Name	Forgot Password
ID	3
Description	The forgot password function is used so when users forget their password they can reset it. They will get sent a onetime code via email which they then can use to reset their password.
Trigger	User clicks on the “Forgot Password” button on the Login Frame
Actors	User
Pre-Condition	Connection to the internet, user must know their username or email they used for creating their account
Post-Condition	User has a new password they can use to login into their account
Basic flow	
Description	User successfully authenticates using Username or Email
Actions	
1	User enters Username or Email and clicks “send code”
2	User enters code that got sent to them via email
3	User enters new password, password gets hashed and then pushed into the database
4	User gets redirected to the login frame
Alternative Flow	A
Description	Code not correct
Actions	
1	User enters username or email and clicks “send code”
2	Controller checks if codes are the same
3	Code is not correct
4	Error Message displayed
5	User can reenter the code
Alternative Flow	B
Description	Username or Email wasn’t found
Actions	
1	User enters Username or Email and clicks “send code”
2	DatabaseHandler couldn’t find entry for username or email
3	Error Message displayed
4	User can reenter the credentials
Alternative Flow	C
Description	Password doesn’t match or not complicit with our password rules
Actions	
1	User enters Username or Email and clicks “send code”
2	User enters code that got sent to them via email
3	User enters new password and password confirmation. They don’t match or don’t follow our password rules
4	Error Message displayed
5	User can reenter password

Figure 15: Snowcard-Forgot Password

4.1.3.2 Use Case Diagram

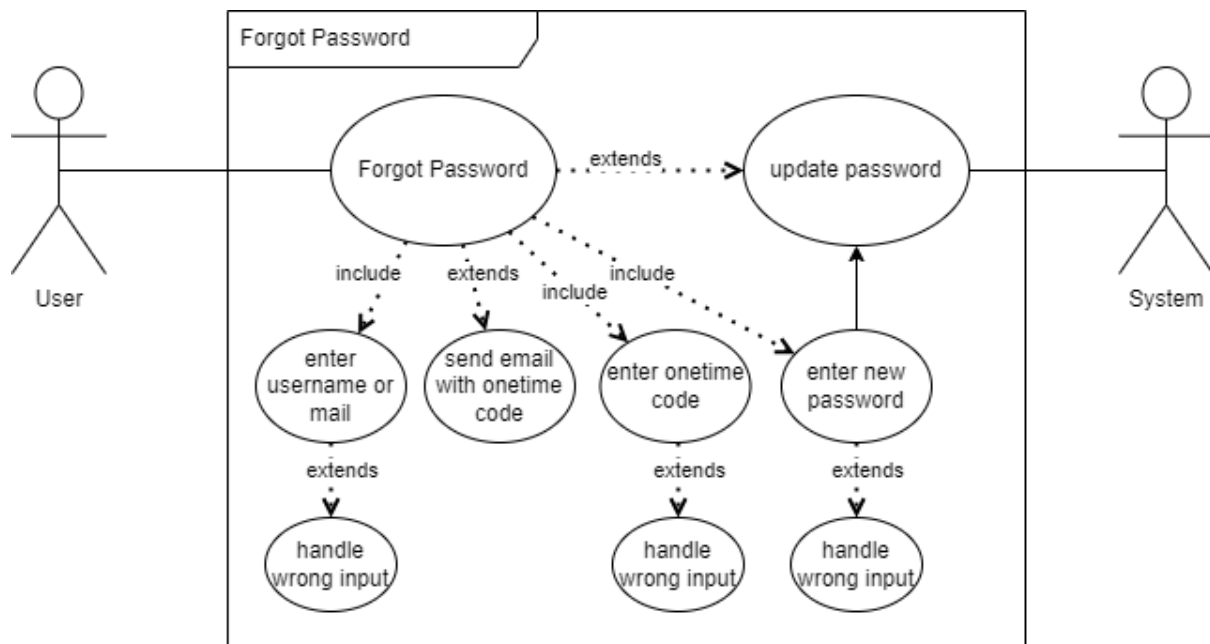
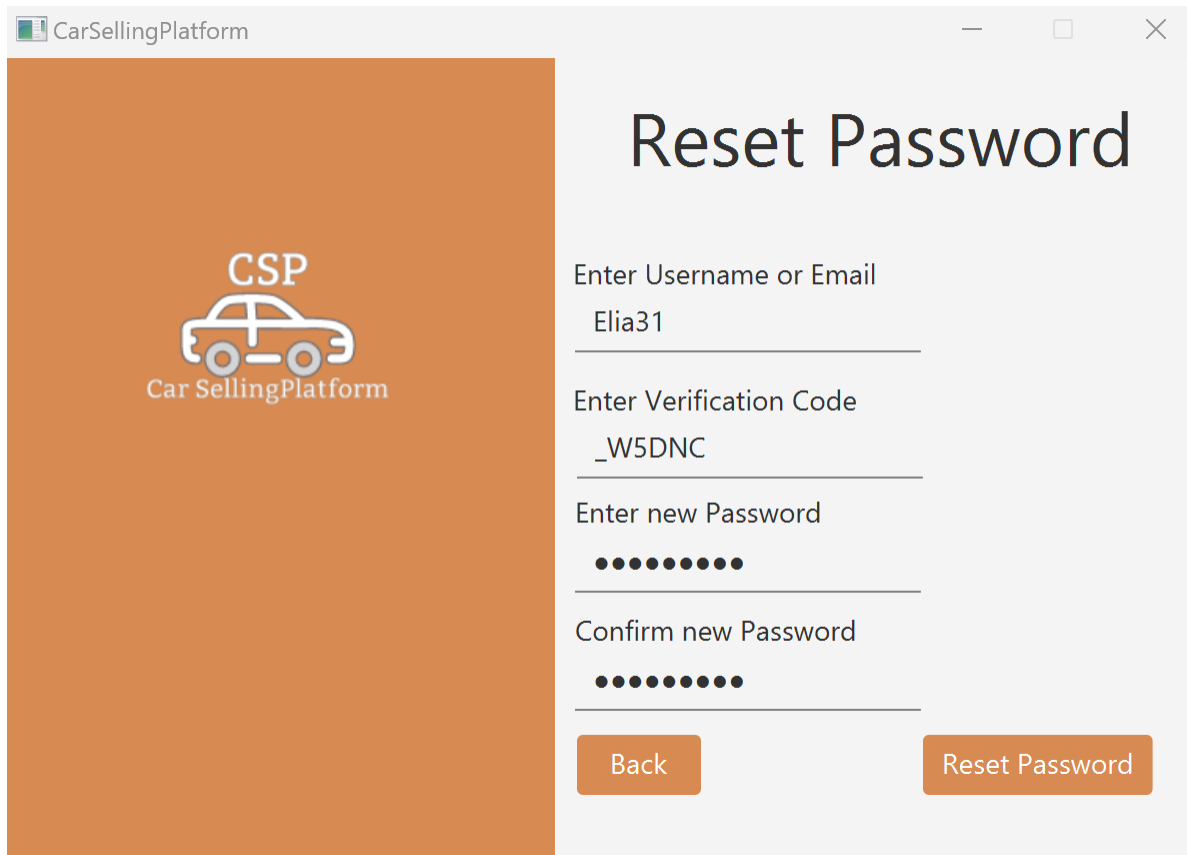


Figure 16: Use Case-Forgot Password

The shown use case diagram shows the basic functionality of the "Forgot Password" feature of our program. When the user clicks on the Forgot Password button on the login frame, he is directed to the Reset Password frame. There he has the possibility to enter either his username or his email. Once he has done this, he will receive an email with a code he must enter to confirm his identity. If the code is correct, he can now think of a new password. The new password will be stored in the database and the user will be redirected to the login frame.

4.1.3.3 User Interface



The screenshot shows a web browser window titled "CarSellingPlatform". The interface is split into two main sections. On the left, there is an orange vertical panel containing the CSP logo, which features a white car icon and the text "CSP" above "Car SellingPlatform". On the right, the background is light gray and contains the heading "Reset Password" in a large, dark font. Below the heading, there are four input fields with labels: "Enter Username or Email" (containing "Elia31"), "Enter Verification Code" (containing "_W5DNC"), "Enter new Password" (with masked dots), and "Confirm new Password" (also with masked dots). At the bottom of this section are two orange buttons: "Back" on the left and "Reset Password" on the right.

Figure 17: User Interface-Reset Password

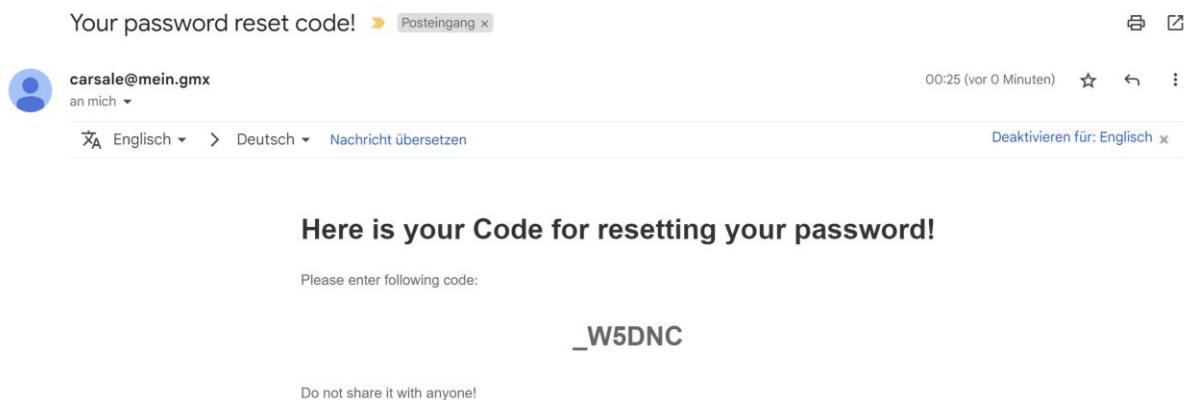


Figure 18: Reset Password-E-Mail

The two images displayed above depict the appearance of the Reset Password Frame and the Reset Password Email. It's worth noting that the code in the Email is entirely unique on each occasion.

4.1.3.4 Sequence Diagram

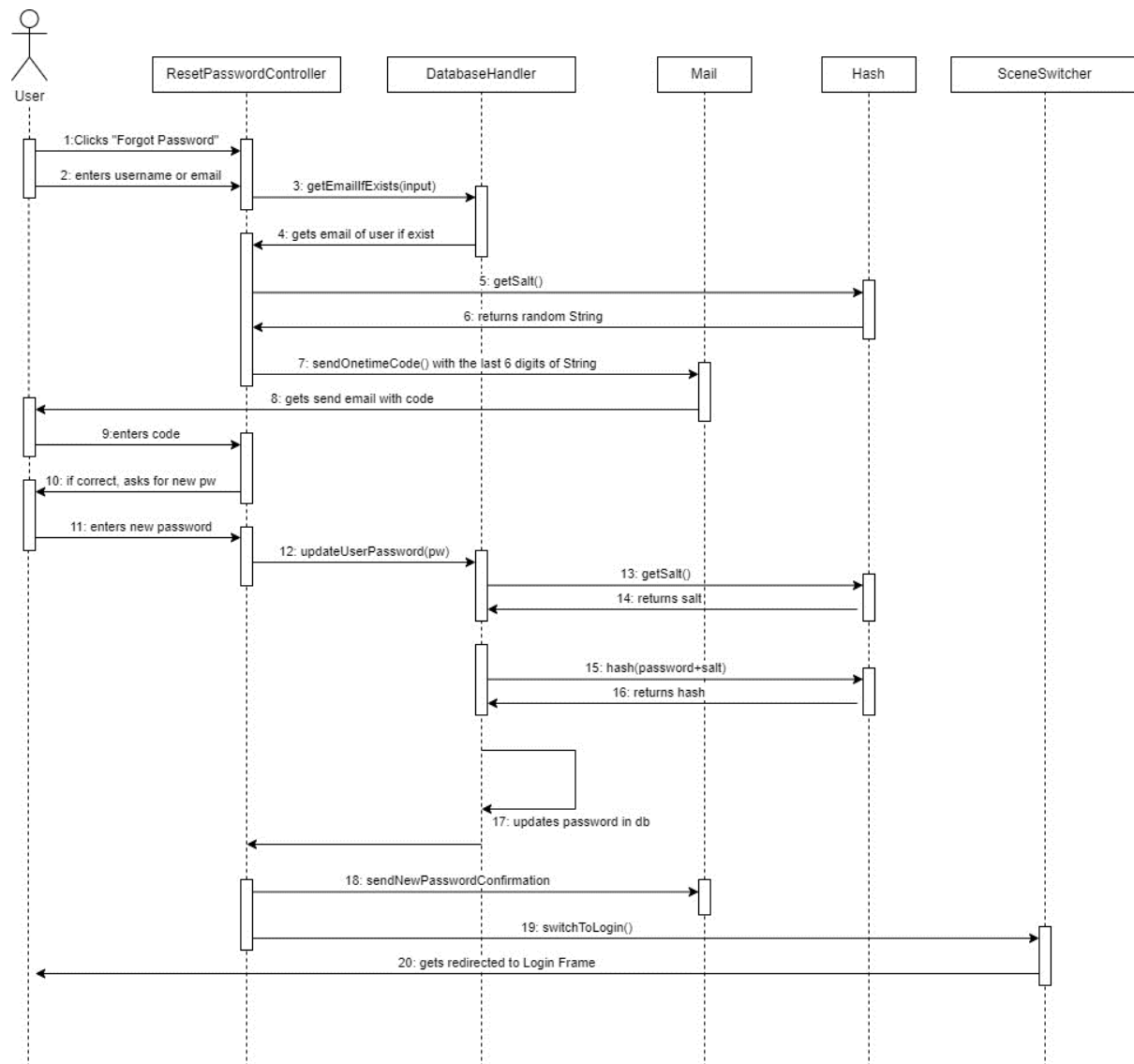


Figure 19: Sequence Diagram-Forgot Password

This sequence diagram illustrates in detail how the "Forgot password" function works in our program and provides a comprehensive overview of the various steps.

The sequence starts when the user clicks on the "Forgot password" option. A new window will then open, asking the user to enter either their username or email address. As soon as the user clicks on "Send Code", the "getEmailIfExist()" function of the "DatabaseHandler" class is used to check whether a user with the entered login data exists.

If a user exists, the "getSalt()" function of the "Hash" class generates a random six-character string. This is then sent using the "Mail" class by displaying a corresponding field where the user must enter the code received via email. Once the entered code matches the previously sent one, two fields are displayed: a password field and a confirmation field for the new password.

Once the user has entered a password that complies with our password policy and matches in both fields, they can click "Reset Password". The "DatabaseHandler" class then uses the "updateUserPassword()" method to store the new password in the database. At the same time, the user receives an email confirming the successful password reset.

Finally, the user is redirected back to the login window using the "SceneSwitcher" class to log in with the updated password.

4.1.3.5 Activity Diagram

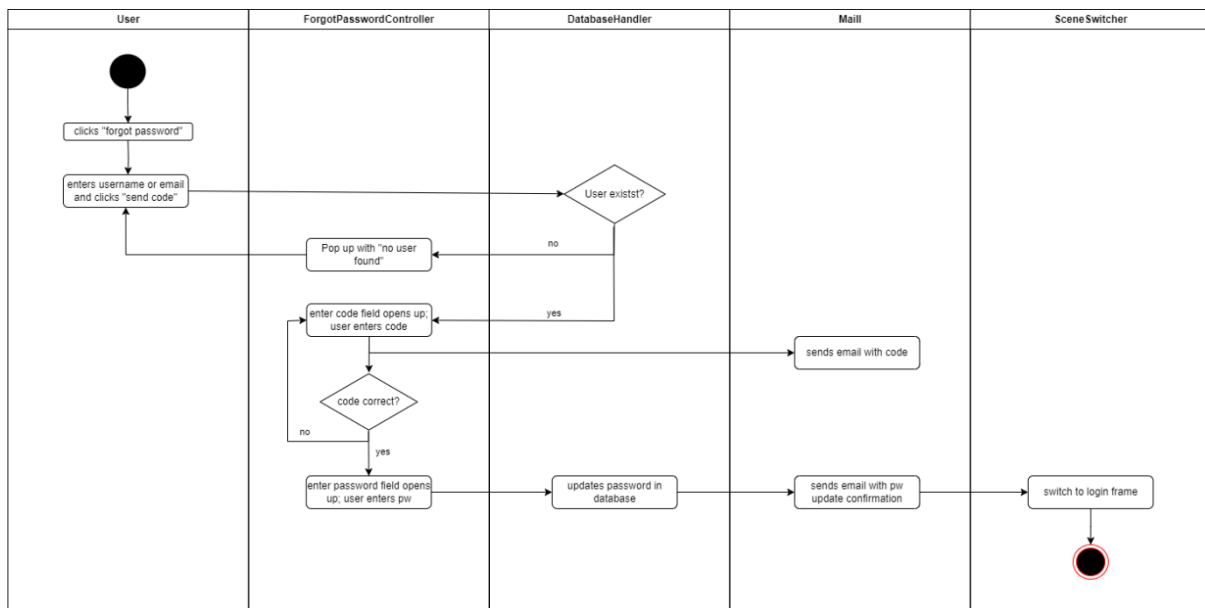


Figure 20: Activity Diagram-Forgot Password

The "Forgot password" functionality of our program starts when a user clicks on the "Forgot password" option in the login frame. In response, the system launches a new window prompting the user to input their username or email address. After the user inputs the requested information, the system verifies if a user associated with the entered credentials exists in the database. Upon successful identification of a user, the system generates a six-digit confirmation code. This code is subsequently dispatched to the email address associated with the user's profile. Concurrently, the user is notified that a code has been sent to their registered email and is asked to input the received code.

The user proceeds by entering the received code in the designated field. The system then validates whether the inputted code aligns with the code initially dispatched. In the event of a match, the system reveals two input fields for the user to set a new password.

The user is then required to input a new password and reconfirm it in the second field. The system proceeds to validate whether the newly inputted password adheres to the password policy and whether the password entries in both fields coincide. If the newly set password is valid and matches the reconfirmation, the system updates the password in the database.

To conclude the password reset process, the user receives a confirmation email notifying them of the successful password reset. The system then redirects the user back to the login window, enabling them to log in using their newly updated password.

4.1.4 Dashboard

4.1.4.1 Snowcard Dashboard

Name	Dashboard
ID	4
Description	The dashboard is the main part of the application, from here the user can search the database for a listing and he can get to any functionality the application provides
Trigger	User click on the login button on the login frame
Actors	User
Pre-Condition	User needs an account; User has to be logged in
Post-Condition	User can view all listings
Basic flow	
Description	All listings are visible
Actions	
1	User sets no filters
2	Dashboard Controller searches the database
3	Dashboard Controller displays all available listings
Alternative Flow	A
Description	Only the listings that fulfill the search criteria are visible
Actions	
1	User sets the filters he wants
2	Dashboard Controller applies the filters
3	Dashboard Controller checks if there are listings that match the criteria Dashboard Controller displays all listings that match the filters
Alternative Flow	B
Description	No listings found that match the criteria
Actions	
1	User sets the filters
2	Dashboard Controller applies the filters
3	Dashboard Controller checks if there are listings that match the criteria
4	Dashboard Controller tells the user that there are no listings available matching the given criteria

Figure 21: Snowcard-Dashboard

4.1.4.2 Use Case Diagram

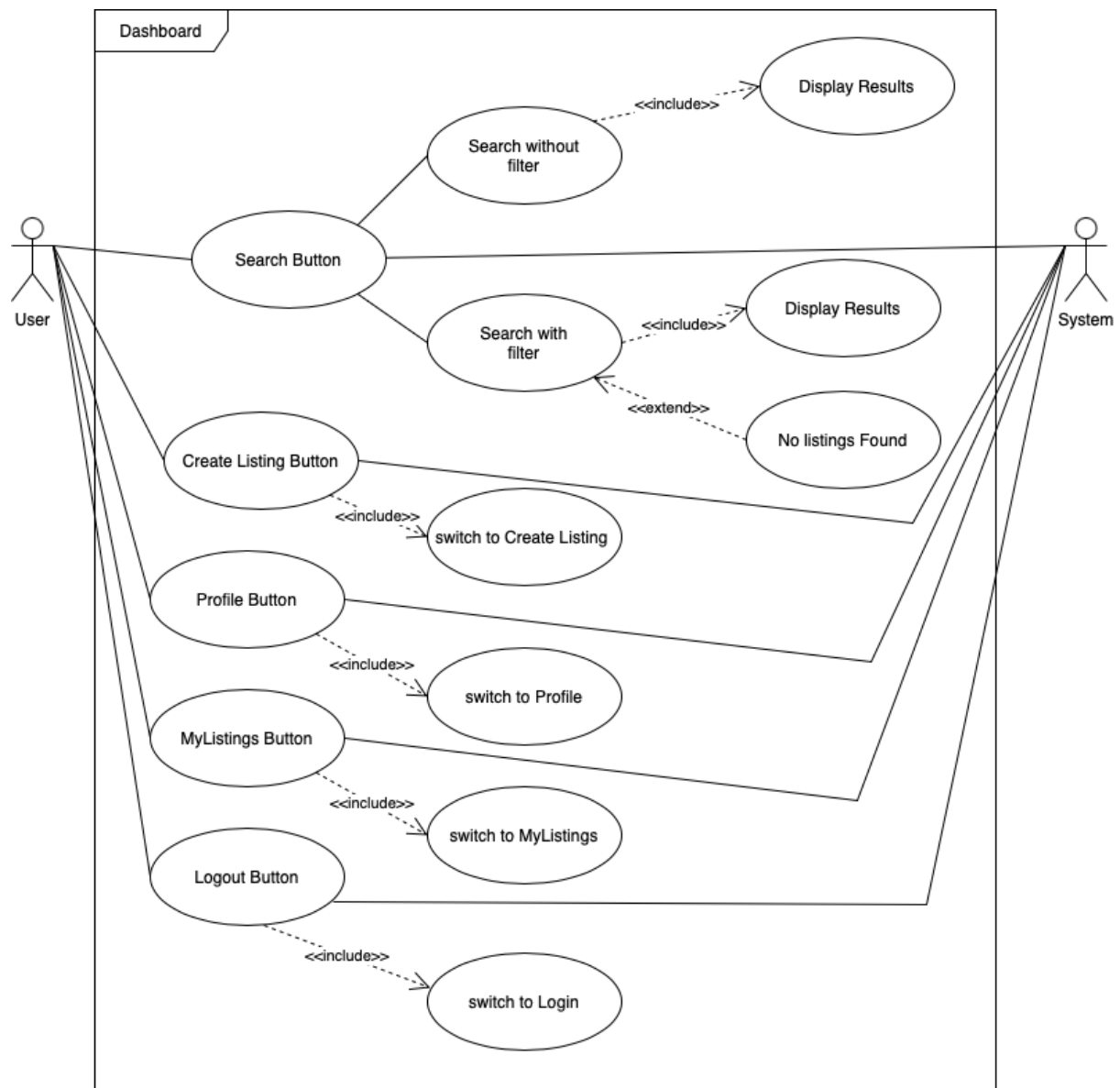


Figure 22: Use Case-Dashboard

This use case diagram shows how the dashboard of the platform works. On the dashboard the user has the option to search the database for available listings he can do that either without any filters and just browse through all available listings or he can set filters and get specific results.

If the user clicks on the CreateListing button he will be redirected to the create listing frame. The user can also click on the Profile button, and he will be able to see the frame with all his personal information. If he clicks on the MyListings button he will get a list of all his listings. If the wants to logout he clicks on the Logout button and he will be redirected to the login screen.

4.1.4.3 User Interface

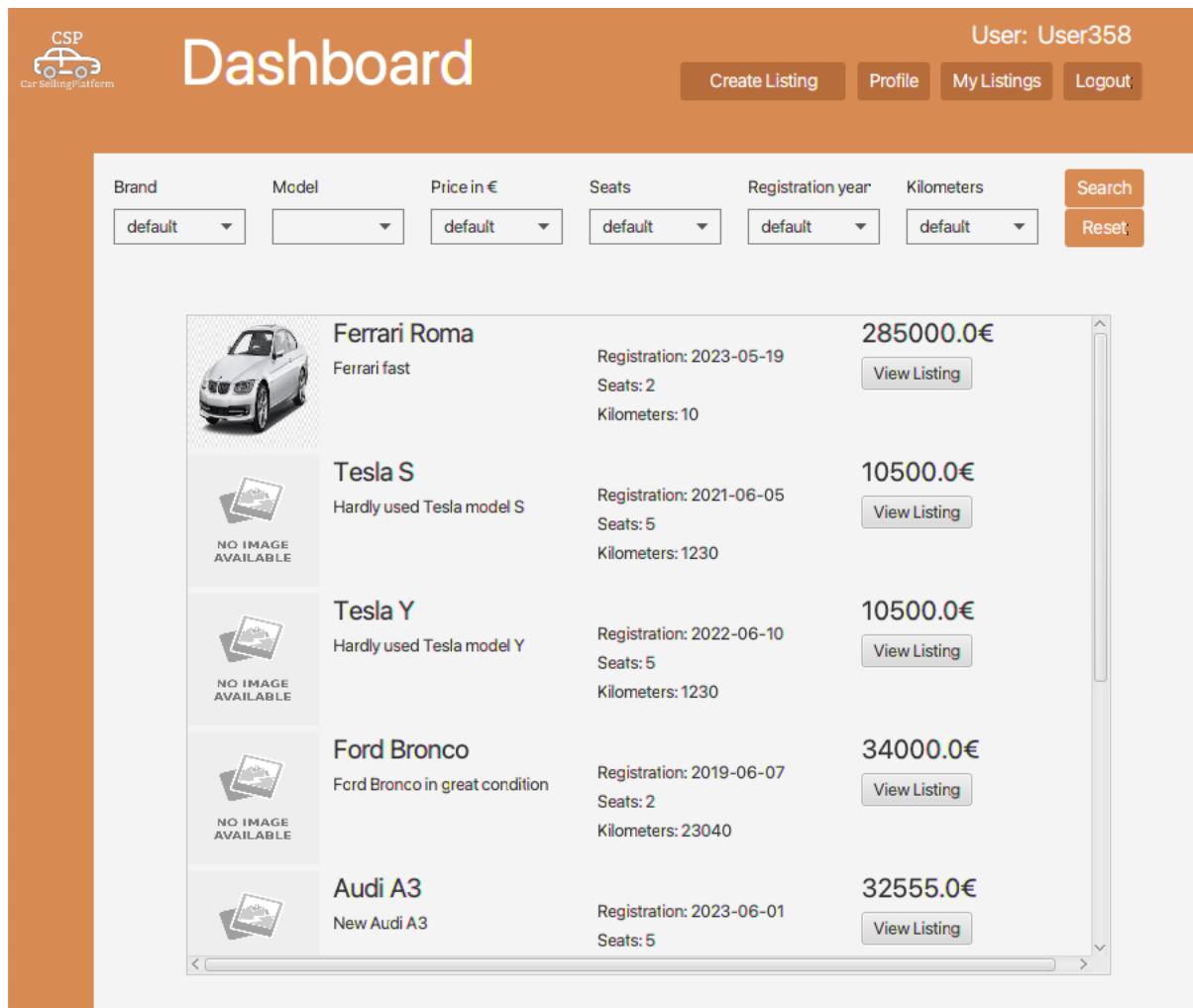


Figure 23: User Interface-Dashboard

In this picture the user interface of the dashboard can be seen. At the top right the username of the user is displayed. Below that there are four buttons. The Create Listing button send the user to another frame there he can create a new listing. The Profile button send the user to another frame there he can view or change his profile information. With the My Listings button the user can look at all his listings. The Logout button logs the user out of his account.

The main part consists of a list with all available listings. Each listing consists of an image, the name of the listing and the description. Also, the user sees the price of each listing and some relevant data. The user can scroll through them and if he wants to see more details of a specific listing, he can press the view Listing button.

Above the list of listings, there are six selection boxes there the user can set filters so the search will be specific to the user's needs. An important detail is, that the user can only select a model if he selected a brand first. On the right side of the selection boxes, there are two more buttons.

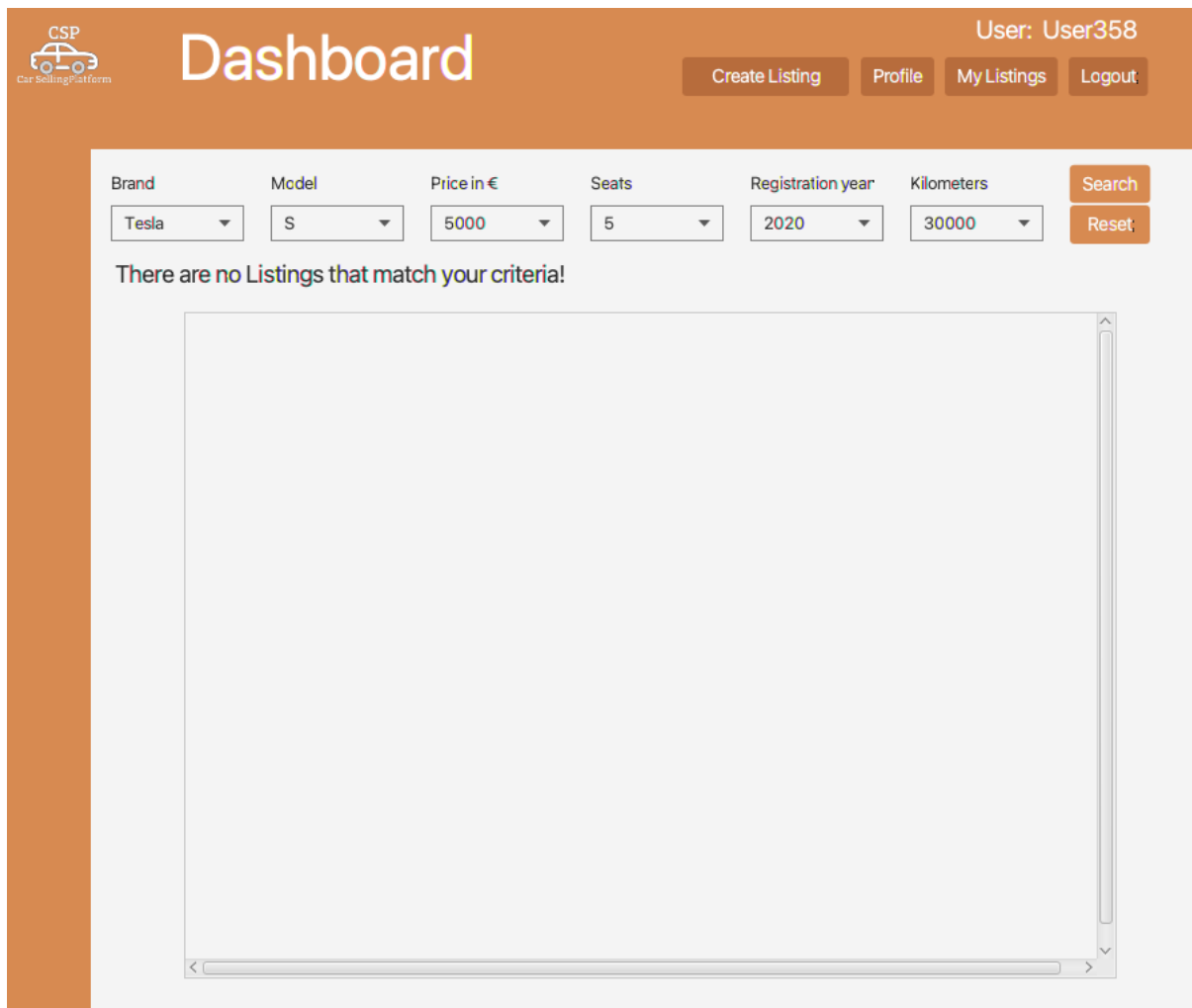
The Search button and the Reset button. The Search button applies the selections to the search and displays the modified results. The Reset button resets all filters and displays all available listings without any filters.

The screenshot shows the CSP (Car Selling Platform) Dashboard. The header is orange and contains the CSP logo, the word "Dashboard", and the user name "User: User358". Below the header, there are several buttons: "Create Listing", "Profile", "My Listings", and "Logout". The main content area has a search filter section with the following fields: Brand (Tesla), Model (S), Price in € (15000), Seats (5), Registration year (2020), and Kilometers (40000). There are "Search" and "Reset" buttons next to these fields. Below the filter section, a listing for a "Tesla S" is displayed. The listing includes a placeholder image with the text "NO IMAGE AVAILABLE", the title "Tesla S", the description "Hardly used Tesla model S", the registration date "2021-06-05", the number of seats "5", the kilometers "1230", and the price "10500.0€". A "View Listing" button is located next to the price.

Brand	Model	Price in €	Seats	Registration year	Kilometers
Tesla	S	15000	5	2020	40000

Image	Title	Description	Registration	Seats	Kilometers	Price	Action
NO IMAGE AVAILABLE	Tesla S	Hardly used Tesla model S	2021-06-05	5	1230	10500.0€	View Listing

Figure 24: User Interface-Dashboard with filter



The screenshot shows a web application interface for a car selling platform. The top navigation bar is orange and contains the CSP logo (Car Selling Platform) on the left, the word "Dashboard" in large white text in the center, and the user's name "User: User358" on the right. Below the user name are four buttons: "Create Listing", "Profile", "My Listings", and "Logout". Below the navigation bar is a search filter section with six dropdown menus: "Brand" (set to "Tesla"), "Model" (set to "S"), "Price in €" (set to "5000"), "Seats" (set to "5"), "Registration year" (set to "2020"), and "Kilometers" (set to "30000"). To the right of these filters are two buttons: "Search" and "Reset". Below the filters, a message states: "There are no Listings that match your criteria!". Below this message is a large, empty rectangular area with a scrollbar on the right side, indicating where search results would typically be displayed.

Figure 25: User Interface-Dashboard no result

In the two pictures above the dashboard is shown again, but this time the filters have been set. In the first picture only one listing is shown because it is the only listing that matches the criteria. In the second picture there are no listings that match the criteria, so there is just a text that tells the user that there are no listings with these filters.

4.1.4.4 Activity Diagram

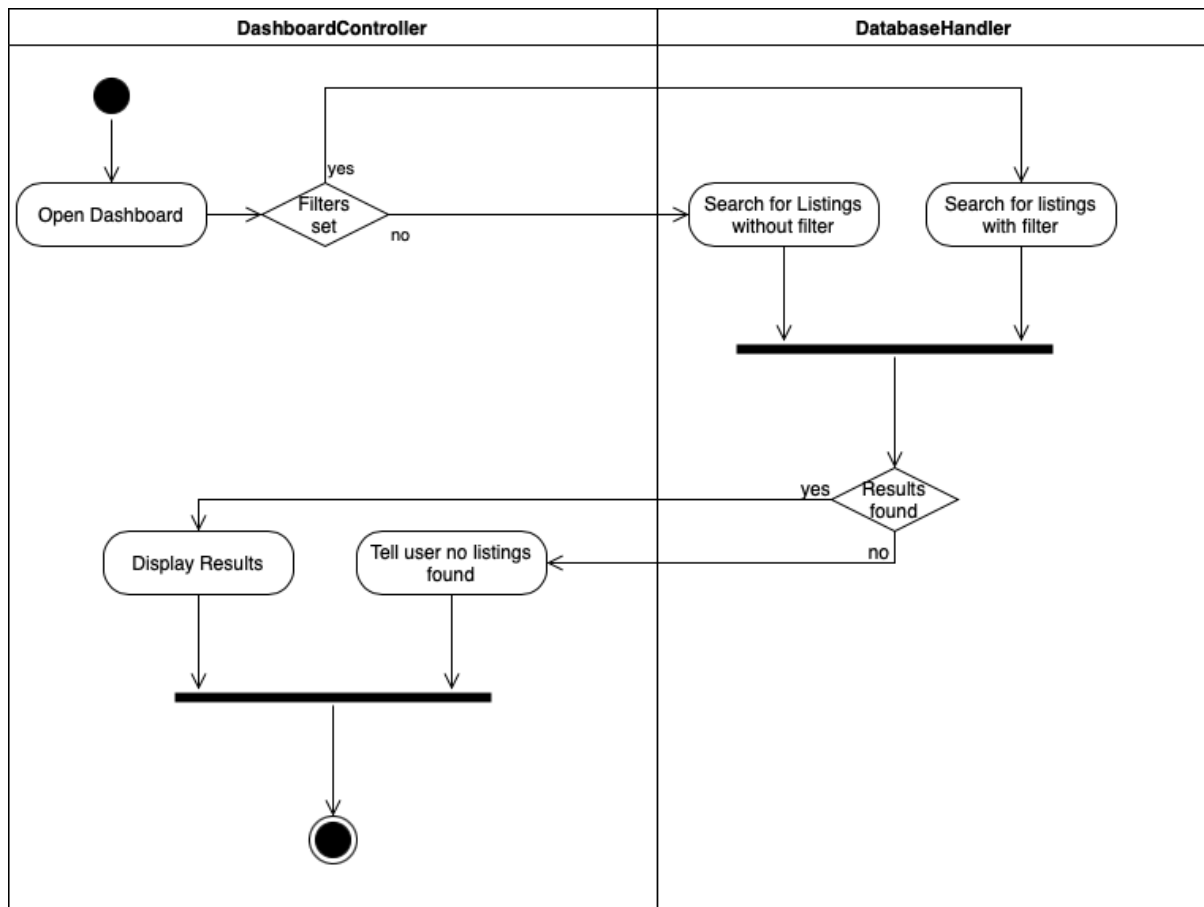


Figure 26: Activity Diagram-Dashboard

The activity diagram provides an overview of the search process in the car selling platform. It details the flow of operations within search method the **DashboardController** class.

After the dashboard is open, the user has two choices. He can either search the whole database for all available listings without any contains or he can set filters and gets a only a subset of listings that match his filter choices.

If the user decides to use the filter option, there are again two possible outcomes. In the case the database contains one or more listings that match his criteria, the dashboard controller will just display the resulting listings. In the other case there are no listings in the database that match the filter selections. If that is the case the dashboard controller will not display any listings and will tell the user with a message that there are no listings for his preferences.

4.1.4.5 Sequence Diagram

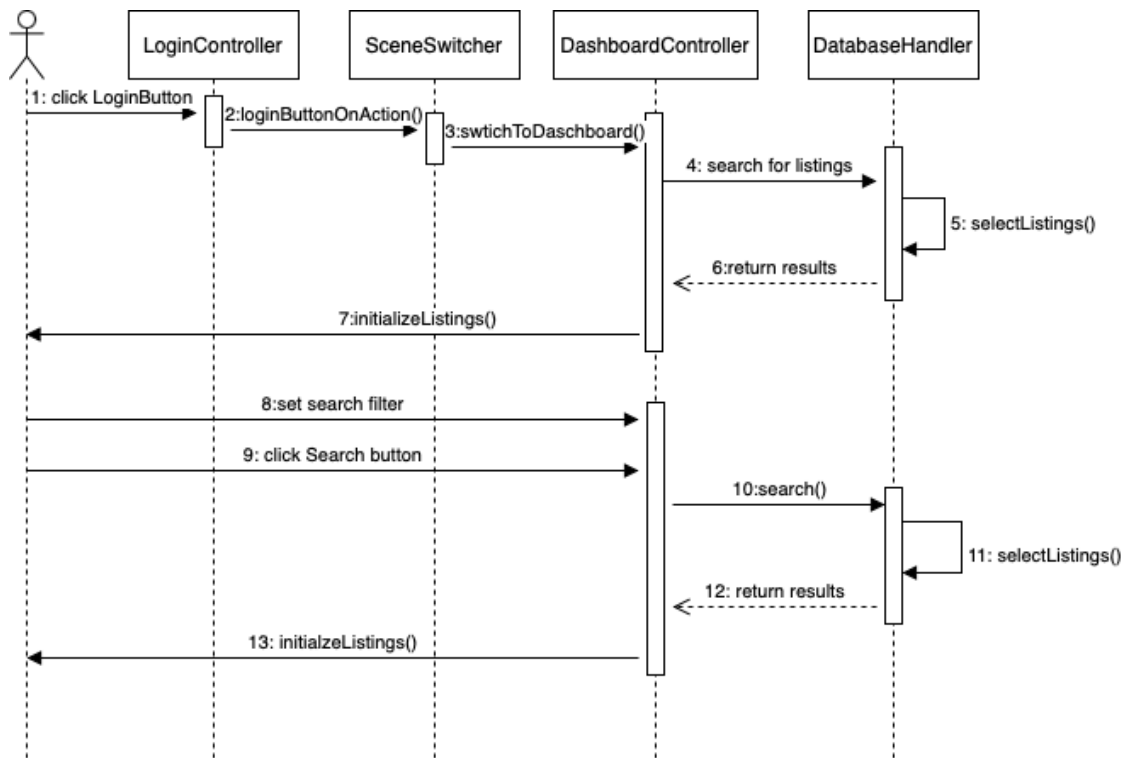


Figure 27: Sequence Diagram-Dashboard

The sequence diagram gives a detailed overview of the search process in the car selling platform. It shows the interactions between the different controller classes needed to complete the search process.

It starts with the user pressing the Login button on the login screen. After that the LoginController will tell the SceneSwitcher to redirect the user to the dashboard and the DashboardController.

For the default display of all listings the DashboardController tell the DatabaseHandler to search for all listings in the database. The DatabaseHandler uses the selectListings method to get the results and returns them back to the DashboardController. The DashboardController then displays all listings for the user.

If the user wants a search that is specific to his preferences, he sets the available filters and clicks the Search button. The Search button uses the search method from the DashboardController to apply the filters to the search request. The DatabaseHandler uses again the selectListings method to get the corresponding results and returns them to the DashboardController. Now the DashboardController uses the initializeListings method to display the results to the user.

4.1.5.3 User Interface

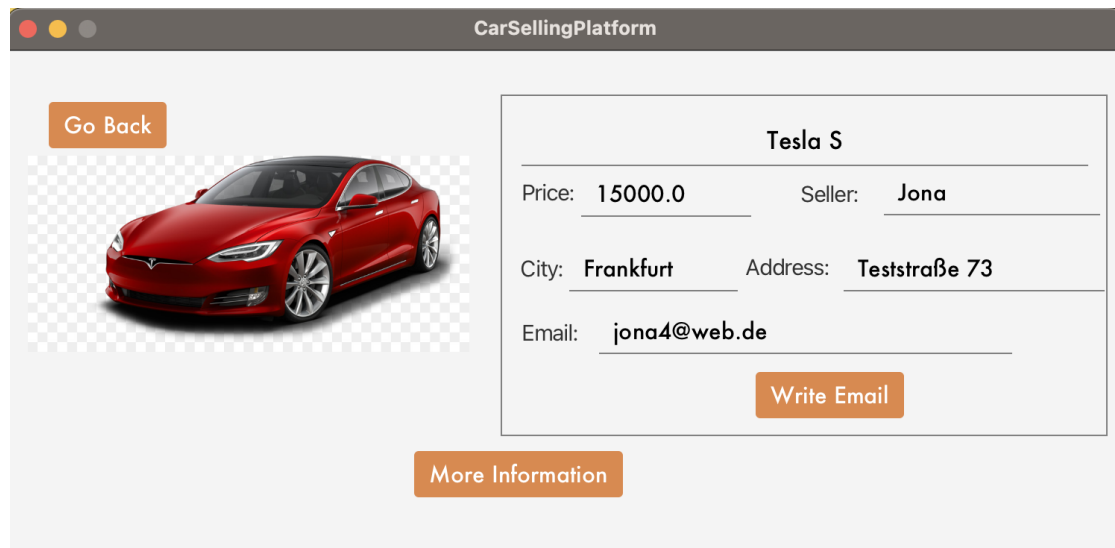


Figure 30: User Interface- Listing View

After selecting a listing from the dashboard, the user is redirected to this window. There all the general information about the listing is displayed.

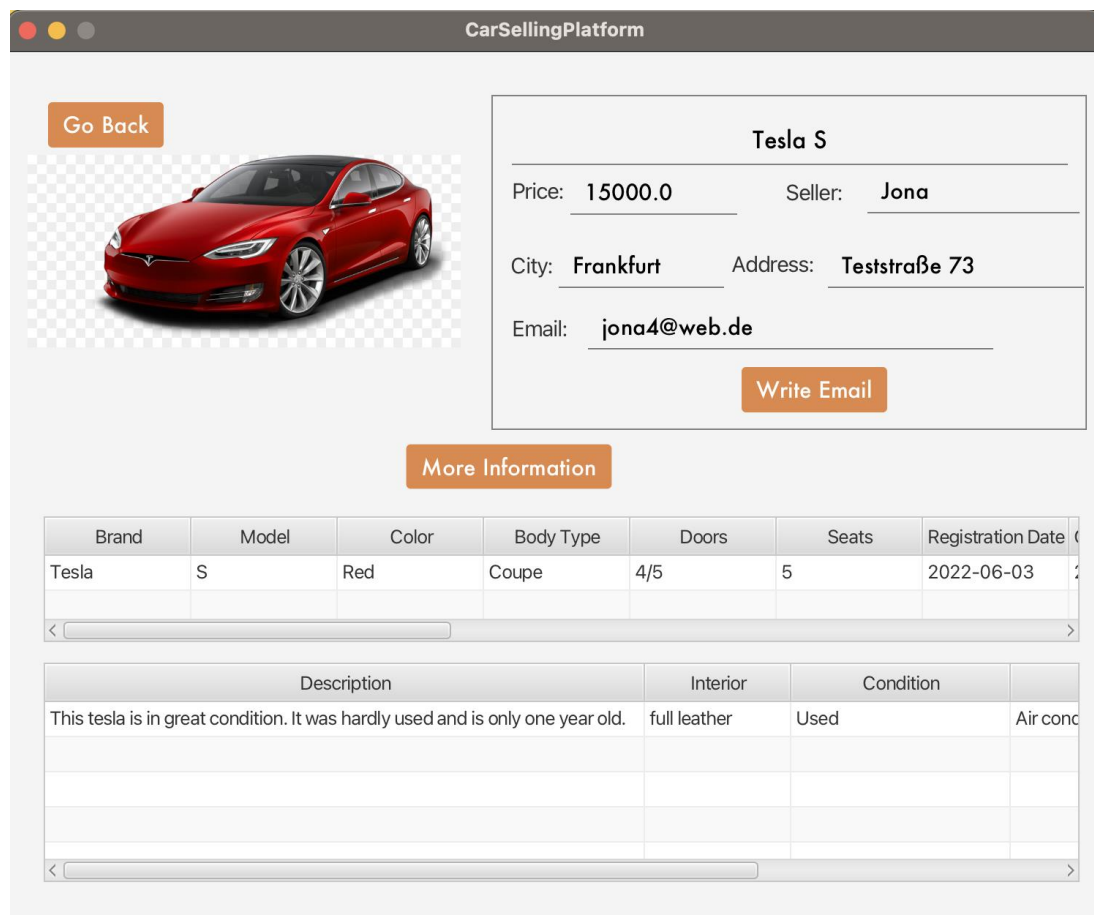


Figure 31: User Interface-Listing View extend Information

The user clicked on the extend Information button. The user gets displayed with all information about the listing. The information is divided into general and technical information tables.

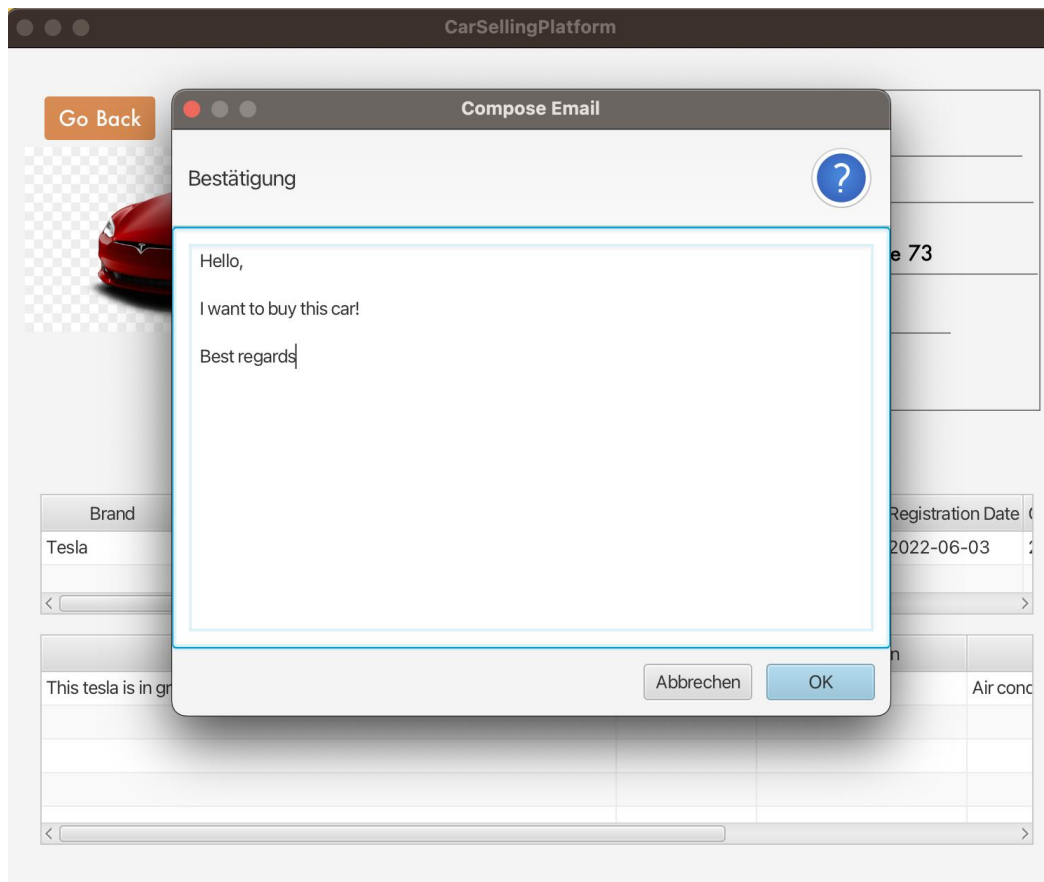


Figure 32: User Interface-Listing View E-Mail-Text Field

After pressing the write email button, the text area is displayed automatically. If the user didn't write anything into the text area an error gets displayed. After entering a message as shown in the picture above the user can press the ok button.

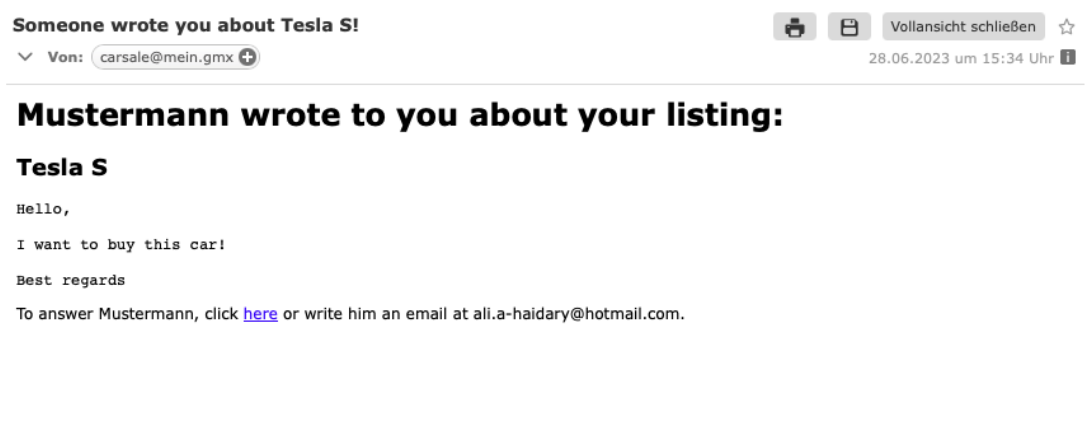


Figure 33: Listing View - E-Mail send to Seller

After that the seller of the car gets an email with the message from the potential buyer. This is also shown in the email snippet above. By clicking the underlined here button the seller can answer the potential buyer.

4.1.5.4 Activity Diagram

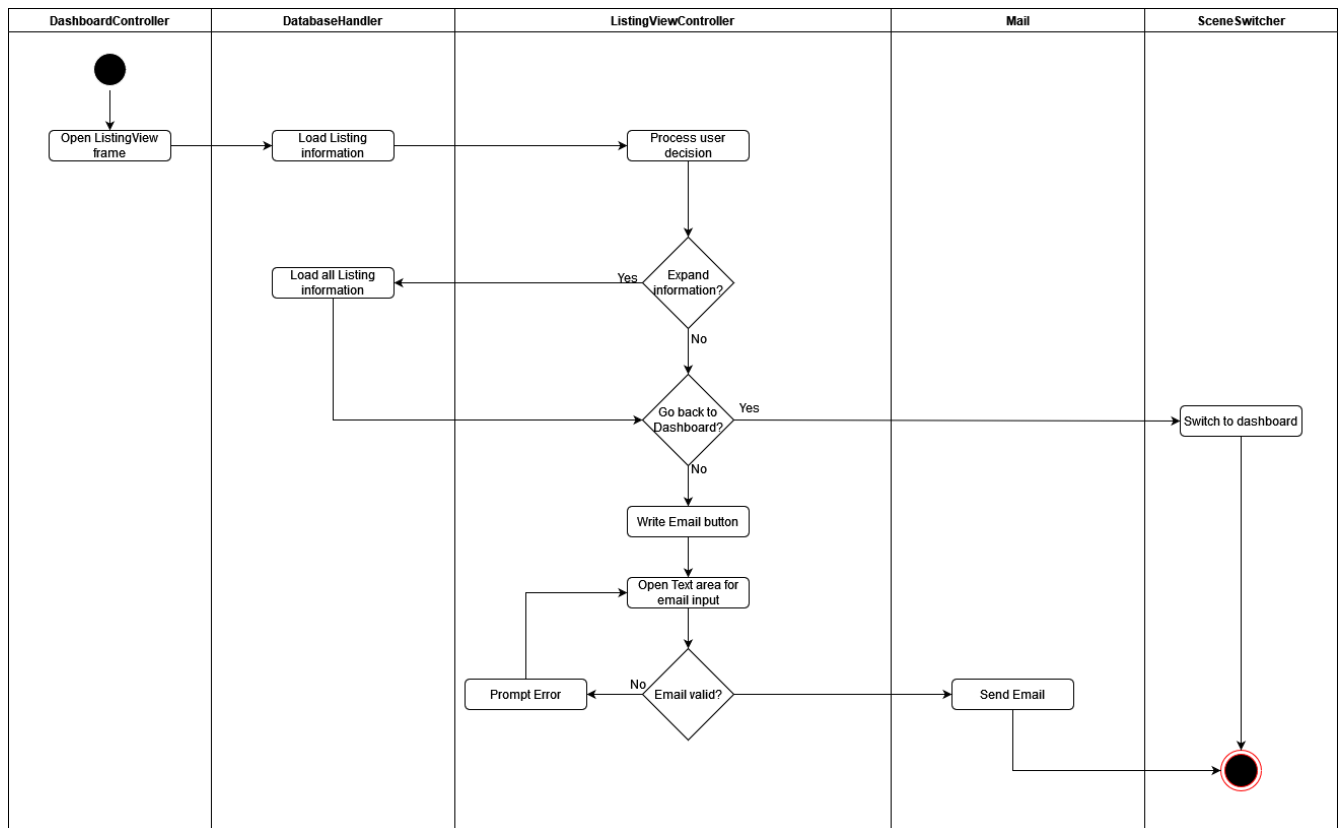


Figure 34: Activity Diagram-Listing View

Upon selecting a listing from the dashboard, the user triggers the activation of the Listing Frame. This action initiates an automatic retrieval of key information from the database, a process managed by the DatabaseHandler class.

Once this information is displayed, the user is presented with several options. They can request additional details about the listing, navigate back to the dashboard, or initiate contact with the seller via email.

If the user wants to write an email, the ListingViewController class conducts a check to ascertain whether any user input has been registered. In the absence of any input, the email sending process is bypassed.

If the user opts to send an email, the process culminates with the dispatch of the email to the seller, marking the end of this user-driven sequence of interactions.

Should the user choose to return to the dashboard, the transition is managed by the Scene Switcher class.

4.1.5.5 Sequence Diagram

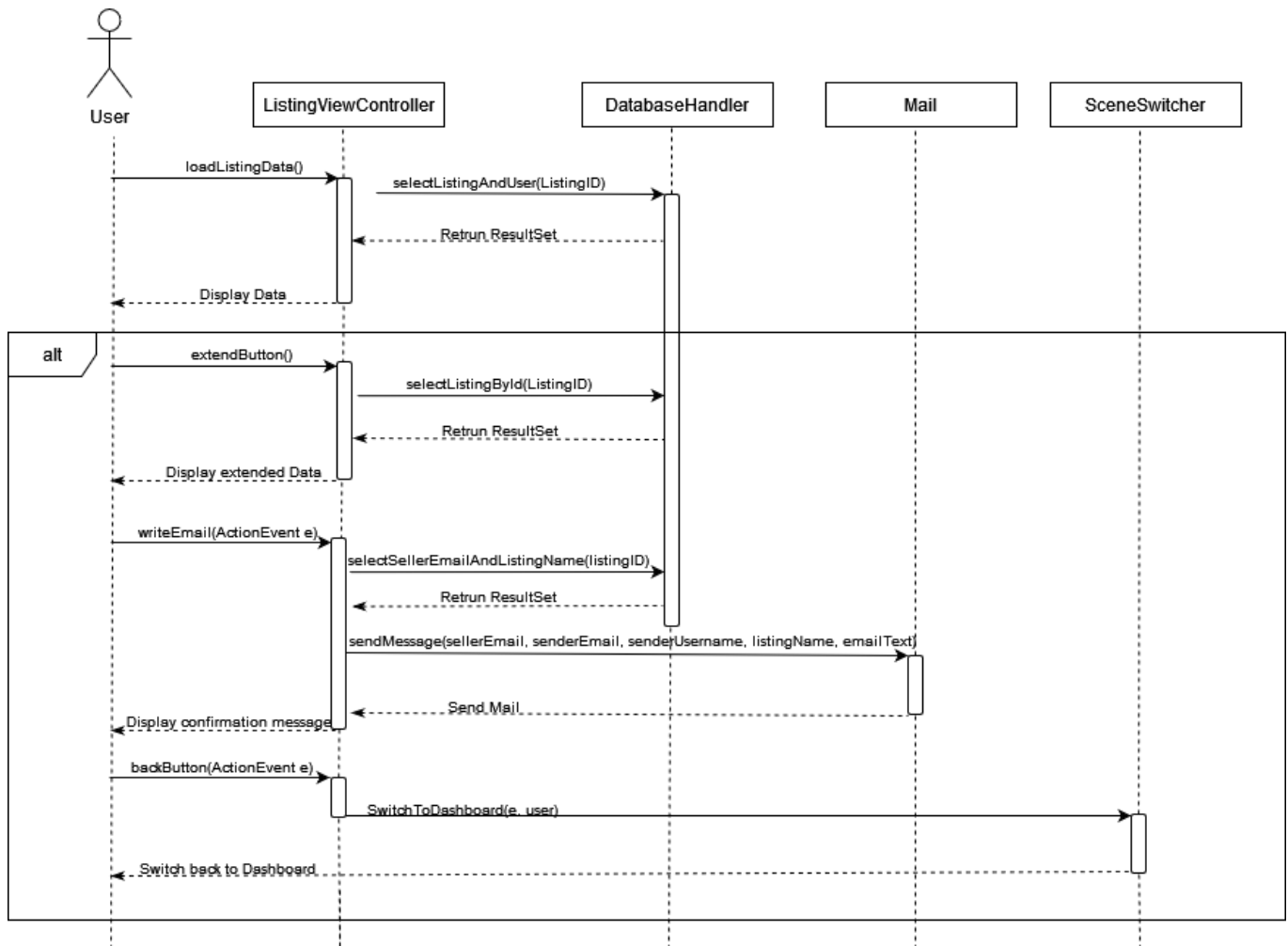


Figure 35: Sequence Diagram-Listing View

In the ListingViewController class, the user initiates a series of interactions. The first one being the invocation of the `loadListingData()`. This method is called after the user clicks on a listing on the dashboard which calls the `selectListingAndUser(listingID)` function from the DatabaseHandler class.

```

1. public ResultSet selectListingAndUser(int listingID) {
2.     try {
3.         String sql = "SELECT Listing.ListingName, Listing.Price, User.Username, User.Email,
4.             User.Address, User.City, Listing.Picture " +
5.             "FROM Listing " +
6.             "INNER JOIN User ON Listing.Username = User.Username " +
7.             "WHERE Listing.ListingID = ?";
8.         PreparedStatement statement = connection.prepareStatement(sql);
9.         statement.setInt(1, listingID);
10.        resultSet = statement.executeQuery();
11.    } catch (Exception e) {
12.        e.printStackTrace();
13.    }
14.    return resultSet;
15. }

```

Figure 36: Select Listing and User-Code

This function fetches both the listing and user data from the database. Once the data is successfully retrieved, it is subsequently displayed to the user.

Following the data display, the user is presented with several options. By clicking on the extend information button the user gets all information about the listing. In this case, the user triggers the `extendButton()` method within the `ListingViewController` class. This method calls the `selectListingById(listingID)` function from the `DatabaseHandler` class, which fetches the extended listing data from the database. The data is then displayed to the user via a two separate tables.

Alternatively, the user may choose the option to return to the dashboard. To facilitate this, the user triggers the `backButton(ActionEvent e)` method by clicking on the Go back button within the `ListingViewController` class. This method then calls the `switchToDashboard(e, user)` function from the `SceneSwitcher` class, which is responsible for transitioning the scene back to the dashboard.

Finally, the user may want to write an email. This action is facilitated by the user triggering the `writeEmail(ActionEvent e)` method via clicking on the write email button.

```

1. public ResultSet selectSellerEmailAndListingName(int listingID) {
2.     String sql = "SELECT User.Email, Listing.ListingName " +
3.         "FROM Listing " +
4.         "INNER JOIN User ON Listing.Username = User.Username " +
5.         "WHERE Listing.ListingID = " + listingID;
6.     Statement stmt;
7.     ResultSet rs = null;
8.     try {
9.         stmt = connection.createStatement();
10.        rs = stmt.executeQuery(sql);
11.    } catch (SQLException ex) {
12.        ex.printStackTrace();
13.    }
14.    return rs;
15. }

```

Figure 37: Select Seller Email and Listing-Code

This method first calls the `selectSellerEmailAndListingName(listingID)` function from the `DatabaseHandler` class to retrieve the seller's email and the name of the listing from the database. Subsequently, the `sendMessage(sellerEmail, senderEmail, senderUsername, listingName, emailText)` method from the `Mail` class is invoked to send the email. Upon successful dispatch of the email, a confirmation message is displayed to the user.

4.1.6 Create Listings

4.1.6.1 Snowcard

Name	Create Listings
ID	6
Description	This requirement allows the users to create car listings and set them for sale.
Trigger	User click on the “Create Listing” on the dashboard frame
Actors	User
Pre-Condition	Connection to the internet, successful logged in, User wants to sell a car
Post-Condition	The users car listing is saved in the database and made viewable for all other users
Basic flow	
Description	User successfully creates a listing with a picture
Actions	
1	User clicks on the “Create Listing” button on the dashboard frame
2	User enters all of the required data to create a listing
3	User clicks on “Add Picture” button and chooses a locally saved picture
4	User clicks on the “CreateListing” button on the CreateListing frame
5	CreateListingController informs the user that the listing got successfully created and redirects to the dashboard
Alternative Flow	A
Description	User successfully creates a Listing without a picture
Actions	
1	User clicks on the “Create Listing” button on the dashboard frame
2	User enters all of the required data to create a listing except the add picture option
3	User clicks on the “CreateListing” button on the CreateListing frame
4	CreateListingController informs the user that the listing got successfully created and redirects to the dashboard
Alternative Flow	B
Description	User leaves at least one information field empty
Actions	
1	User clicks on the “Create Listing” button on the dashboard frame
2	User thinks that all the required information are entered, but forgot to fill out at least one text field
3	User clicks on “Create Listing” button on the Create Listing frame
4	A warning gets displayed that informs the user which information is missing
5	User corrects his mistake and enters valid input on the remaining textfields or choice boxes
6	User clicks on the “CreateListing” button on the CreateListing frame
7	CreateListingController informs the user that the listing got successfully created and redirects to the dashboard
Alternative Flow	C
Description	User tries to enter other characters than numbers on the price, miles, consumption or horsepower input fields
Actions	
1	User clicks on the “Create Listing” button on the dashboard frame
2	User tries to enter invalid characters for an inputfield that only accepts numbers as input
3	A warning gets displayed that informs the user of the mistake
4	User enters valid data and follows the rest of the necessary steps, to create a listing
5	User clicks on the “CreateListing” button on the CreateListing frame
6	CreateListingController informs the user that the listing got successfully created and redirects to the dashboard
Alternative Flow	D
Description	User tries to enter a date for the last general inspection that lays further int the past than the cars first registration
Actions	
1	User clicks on the “Create Listing” button on the dashboard frame
2	User tries to enter a date for the last general inspection that lays further int the past than the cars first registration
3	A warning gets displayed that informs the user of the mistake
4	User enters valid data and follows the rest of the necessary steps, to create a listing
5	User clicks on the “CreateListing” button on the CreateListing frame
6	CreateListingController informs the user that the listing got successfully created and redirects to the dashboard
Alternative Flow	E
Description	User tries to sell a car as “new” even though it was pre owned
Actions	
1	User clicks on the “Create Listing” button on the dashboard frame
2	User sets the car condition as “new” and the number of pre-ownersers to higher than 0
3	A warning gets displayed that informs the user of the mistake
4	User enters valid data and follows the rest of the necessary steps, to create a listing
5	User clicks on the “CreateListing” button on the CreateListing frame
6	CreateListingController informs the user that the listing got successfully created and redirects to the dashboard

Figure 38: Snowcard-Create Listing

4.1.6.2 Use Case Diagram

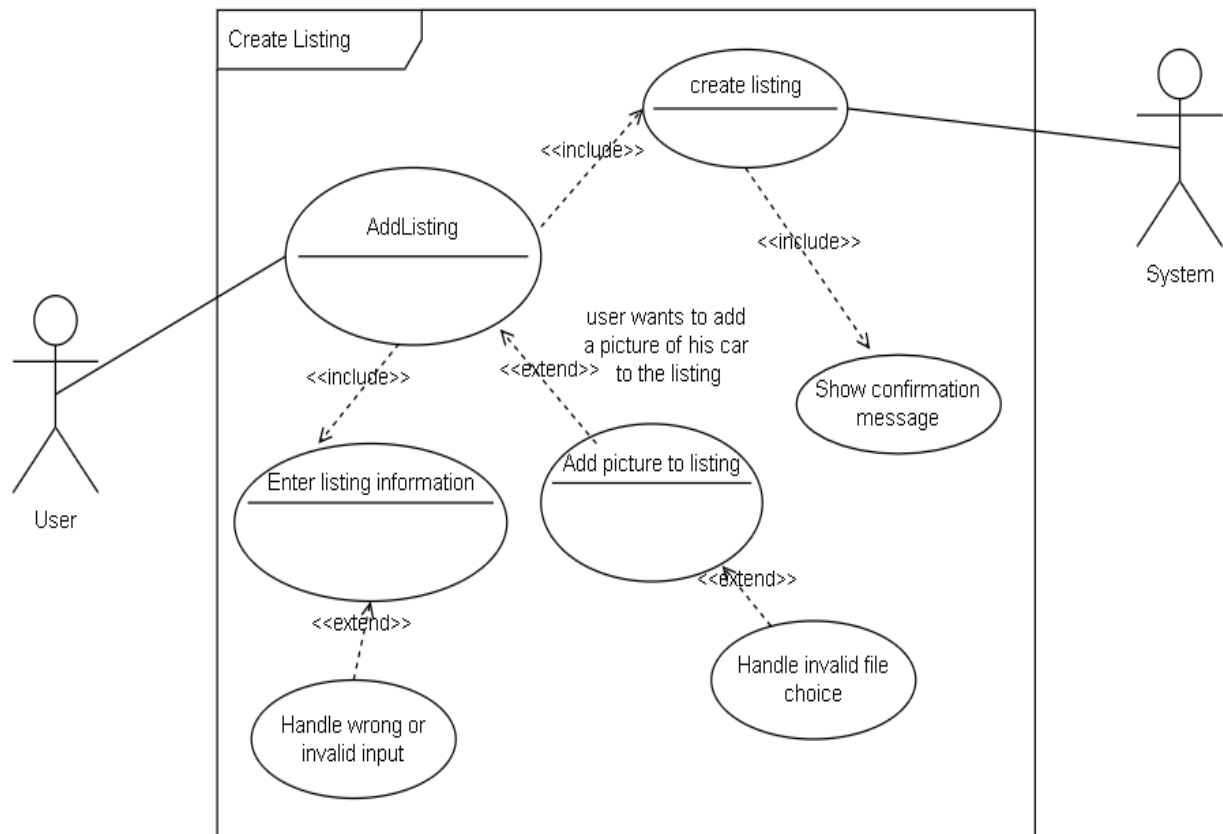



Figure 39: Use Case Diagram-Create Listing

This use case diagram visualizes the interaction between a user and this application/system regarding this requirement in a simple manner. When using the `AddListingController` class which fulfills the “Create Listing”-requirement, the user must enter all information that is required to create a Listing dataset. Wrong input from the user’s side gets handled by the class itself, and if the user feels like it, he can add a picture of his car to the listing. When that is all done, and the input is valid, the system processes all the entered data and uses it so save it as a listing dataset on the application’s database.

4.1.6.3 User Interface

Create a new car listing



NO IMAGE AVAILABLE

Add Picture

Listing Name
Name of listing

Brand
Brand of car

Colour

Interior Material

Last general inspection

Environmental Badge

Performance (in Horse Power)
PS

Price
Price of listing €

Model
Model of car

Number of doors

Air Bags

Number of Pre-Owners

Consumption (l per 100km)
Consumption

Cylinder Capacity

Condition

Body type

Number of seats

Air Conditioning

Miles
Miles

Fuel type

Gearbox

Registration Date

Emission Class

Engine Type

Description

Create Listing

Go Back

Figure 40: User Interface-Create Listing

Here you can see the JavaFX scene that is presented to the user, when trying to create a car listing. On the top left corner of the scene, you can see a placeholder image, which can voluntarily be replaced by a different picture by using the “Add Picture” button below, if the user chooses to do so. Right next to the placeholder image, a collection of text fields, choice boxes and date pickers can be seen, of which each input that is done on one of these, represents an attribute value for the listing that is to be created. The input of the choice boxes is given and cannot lead to any errors, except if the user tries to set the listings condition to “new” while having several pre-owners at the same time. The date pickers work in a way, that the user cannot use dates that lay in the future, and the date of the cars first registration cannot be farther in the past than the date of the car’s first general inspection (cars after their first registration do not get sent to general inspection for the first three years (not to be confused with the initial

registration)). The user can also only enter numerical values for the Performance, Consumption, Miles and Price text field values). And finally on the bottom of the frame you can see 2 remaining buttons of which the upper one, initializes the process of creating a listing dataset and checking the user's input before doing so, and the other button, just directs the user back to the main hub if he changed his mind and does not want to set a vehicle for sale.

4.1.6.4 Activity Diagram

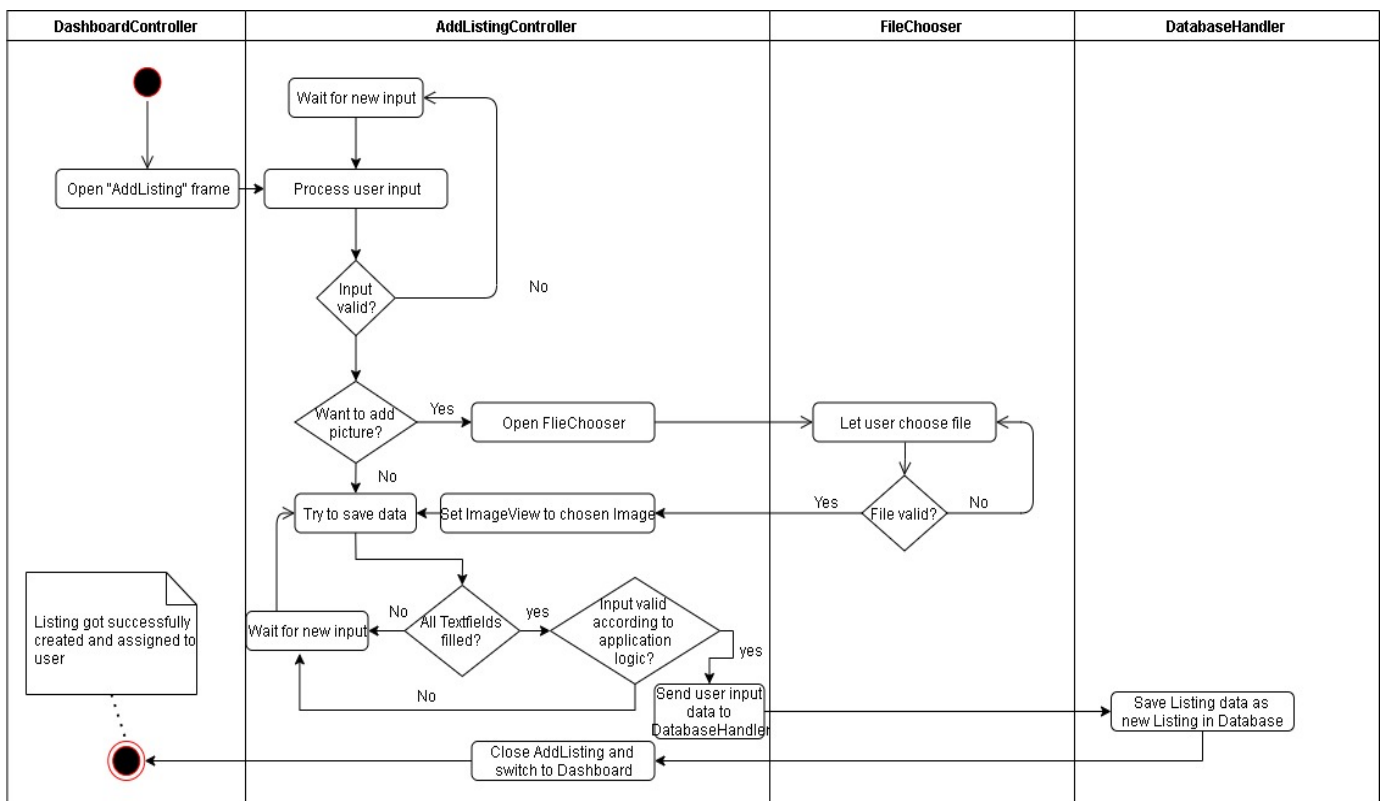


Figure 41: Activity Diagram-Create Listing

This activity diagram visualizes the flow of activities, the requirement of creating a listing creates. The creation flow itself starts with the invocation of a “AddListing”-Frame, which is done here by using the “create Listing” button on the dashboard frame, after that is done, the user gets presented the "create listing"-frame, where he can enter all the required listing information, if the user wants to enter invalid characters to the text fields (like letters to a text field that only accepts numerical characters), the user gets directly interrupted, and told by a warning message, that the input he tried to enter is invalid, and which kind of input is accepted in this case, so the user can then continue and correct his mistake. If the user wants to, he can choose a picture, that should be attached to the listing, if the user wants to add a picture he can use the “add picture” button, which creates an instance of a file chooser, which

then opens the operating system's file explorer (in this case windows), and lets the user choose a locally saved file, in this case, the file chooser only accepts picture files (like .png and .jpg) which do not exceed the size of 16777,215 Kilobytes (because the picture file gets saved as a Medium Blob on the database). If the user chooses an “invalid” file, the system tells him that the chosen file is not supported and gets a chance to choose a different file. If that is done, the frame's placeholder image (ImageView object) gets replaced by the user's chosen picture. Now that the user entered all of the required input, the "AddListingController"-class checks if all of the required information got entered by the user, or if he made one of the mistakes that got discussed before (illogical date or condition choice), if there are any errors, the user has to change some of the given information, and if not, an instance of the "DatabaseHandler"-class gets used to save all the user given input as a listing dataset in the database. After that got successfully done, the user gets an info message that tells him that his listing was successfully created, after that the listing creation frame closes itself and redirects the user to the main hub (dashboard), on which he can probably now see his newly created listing.

4.1.6.5 Sequence Diagram

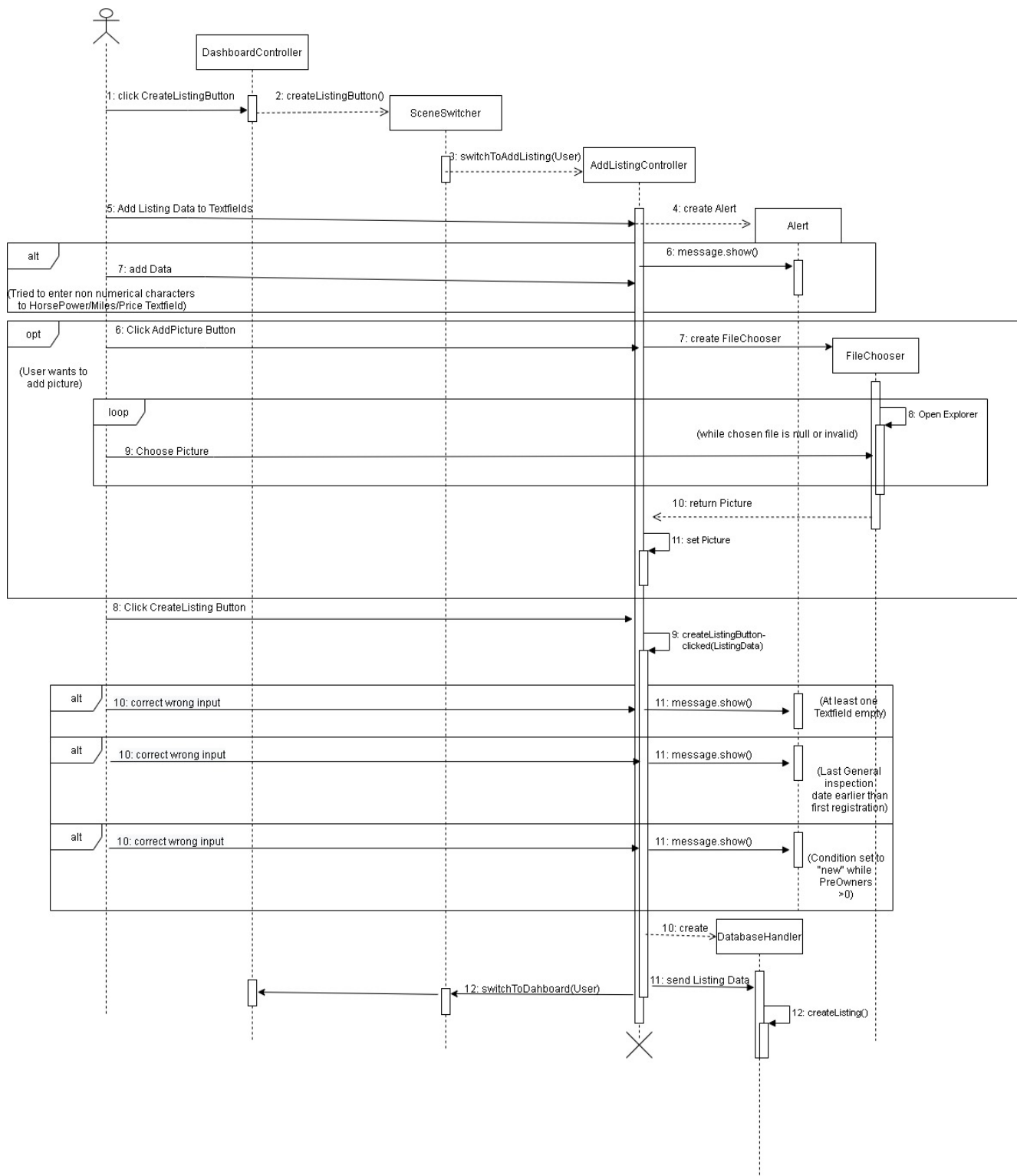


Figure 42: Sequence Diagram-Create Listing

This sequence diagram visualizes what the activity diagram shows, but in a more detailed and “code-close” way.

4.1.7 My Listings

4.1.7.1 Snowcard

Name	My Listings
ID	7
Description	My Listings displays all listings a specific user has created
Trigger	User clicks on the MyListings button on the Dashboard frame
Actors	User
Pre-Condition	User is logged in
Post-Condition	User sees all the listings he created
Basic flow	
Description	All listings are visible
Actions	
1	User clicks MyListings button
2	MyListings Controller searches the database
3	MyListings Controller displays all listings the user has created
Alternative Flow	A
Description	User has no listings
Actions	
1	User clicks MyListings button
2	MyListings Controller searches the database
3	MyListings Controller tells the user that he has no listings

Figure 43: Snowcard-My Listings

4.1.7.2 Use Case Diagram

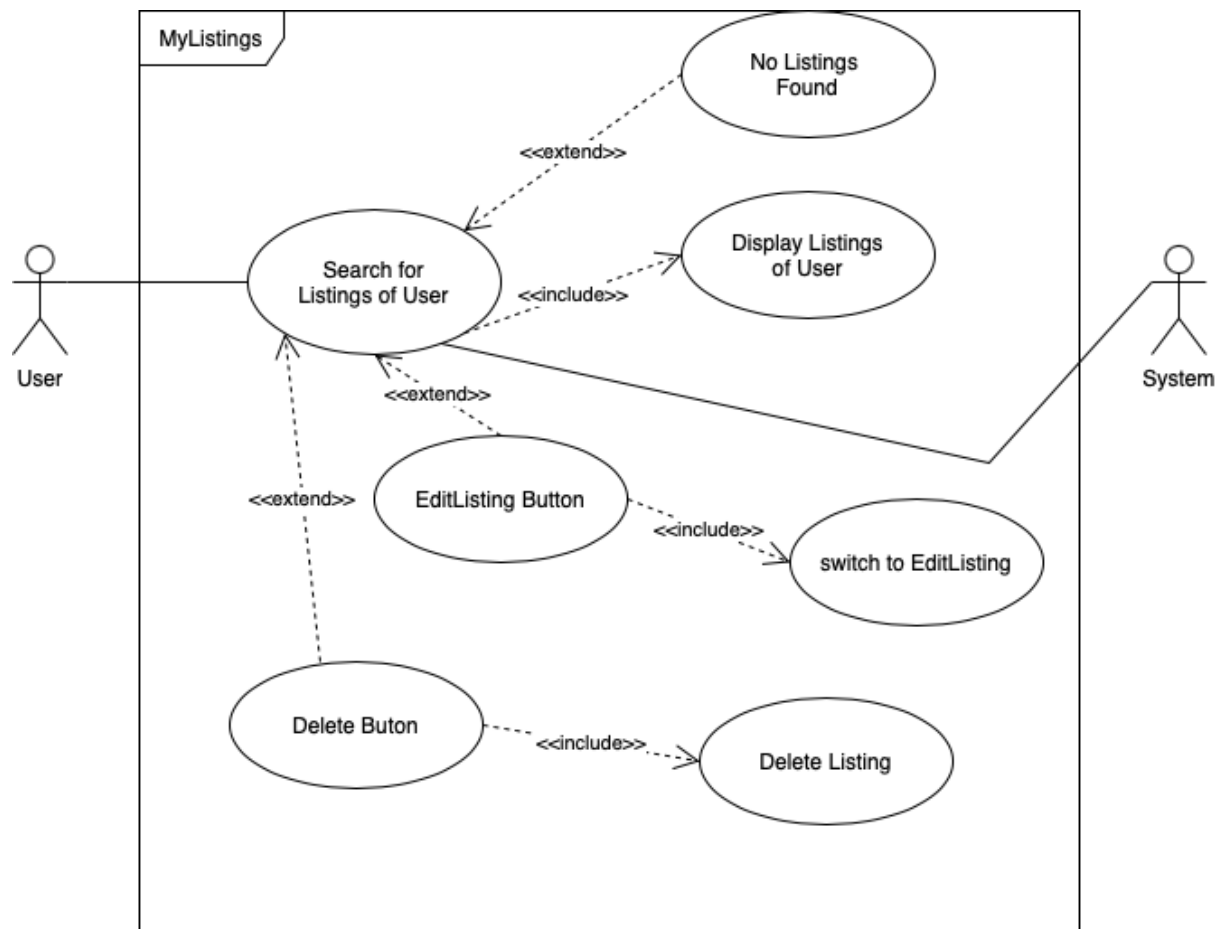


Figure 44: Use Case Diagram-My Listings

This use case diagram shows what happens then the user opens his listings. When the user opens his listings there are two possible displays. In the case the user has already created one or more listings he will see them here and he will be able to edit them.

In case the user does not have a listing at the time the frame will not show any listings but will tell the user that he does not have any listings.

If he wants to edit a listing, he will click on the EditListing button and he will be redirected to the EditListing frame. Also, the user can click on the Delete button this will delete the corresponding listing from the database.

4.1.7.3 User Interface

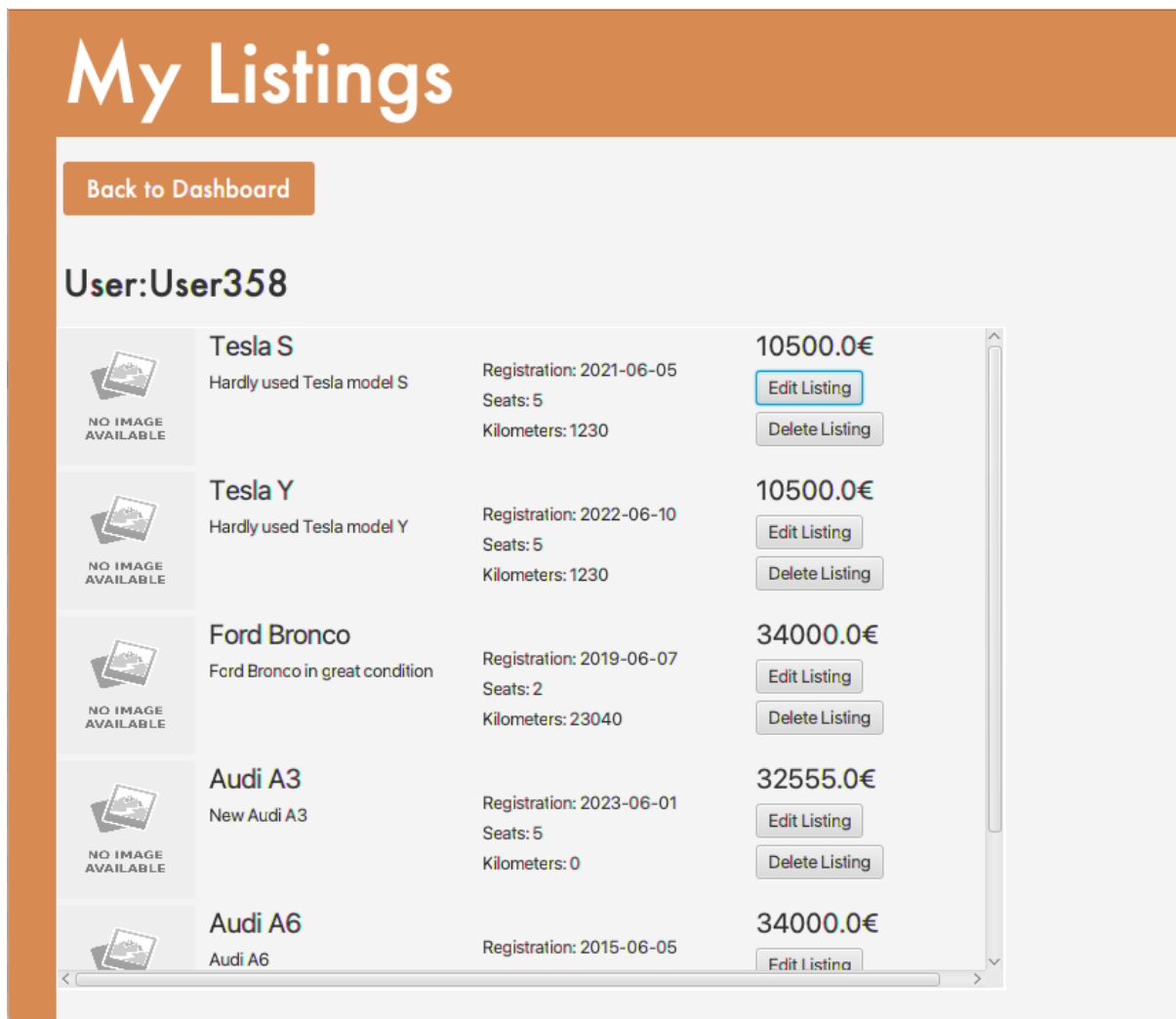


Figure 45: User Interface-My Listings

In the picture the user interface of the MyListings frame can be seen. The with the button on the top left the user can get back to the dashboard. Below that the username is displayed. The main part of this frame is the list with all the listings the user has created.

Each listing has an image, a name, a description, a price, and some other relevant features. Also, each listing has a Edit and delete button. With the Edit button the user can edit a listing and with the delete button the user can delete a listing.

4.1.7.4 Activity Diagram

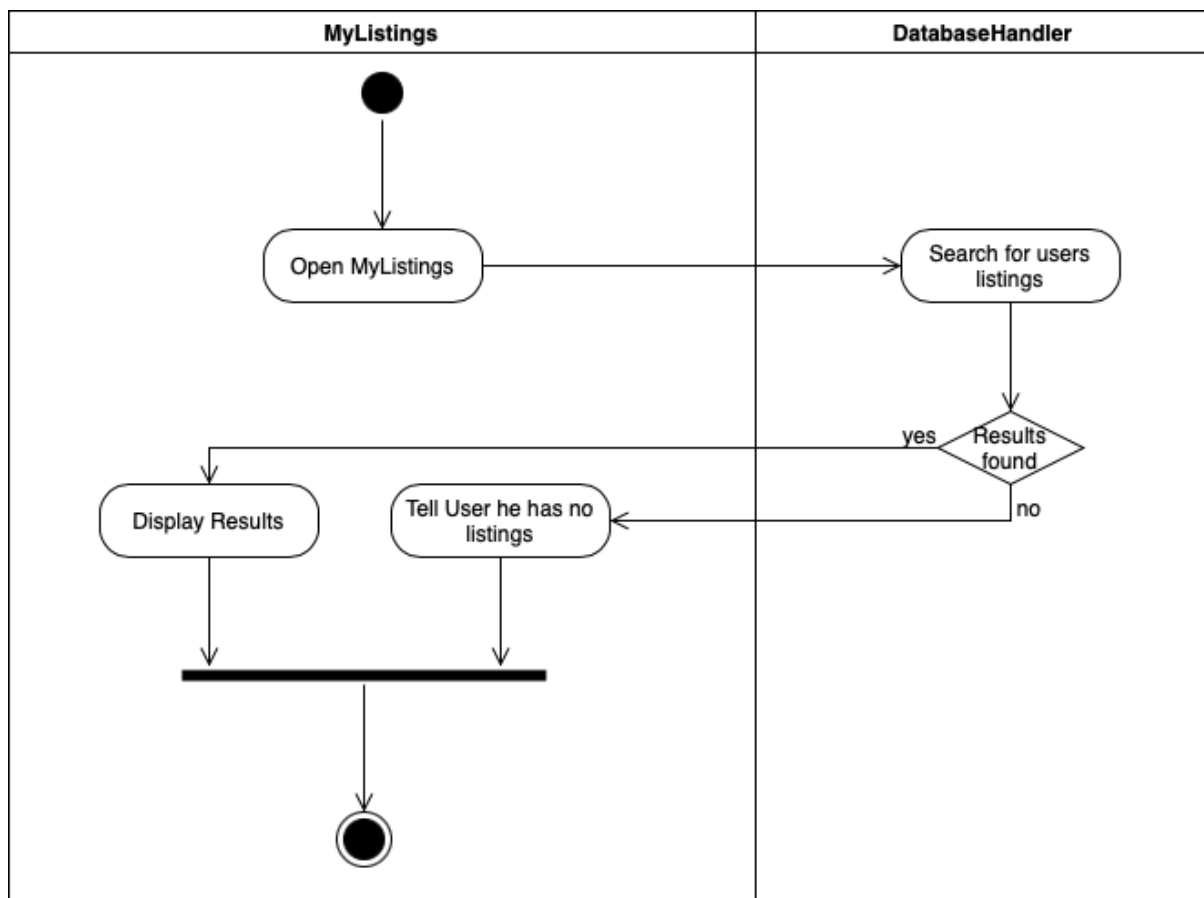


Figure 46: Activity Diagram-My Listings

The activity diagram shows the basic process the MyListings controller goes through to display all listings of a specific user.

Once MyListings is open it tells the database to search for all the listings the user has created. If the user has created at least one listing before it will be displayed on the screen. If the user has not created a listing before there will be no listings displayed and the user gets a message that he has not created any listing.

4.1.7.5 Sequence Diagram

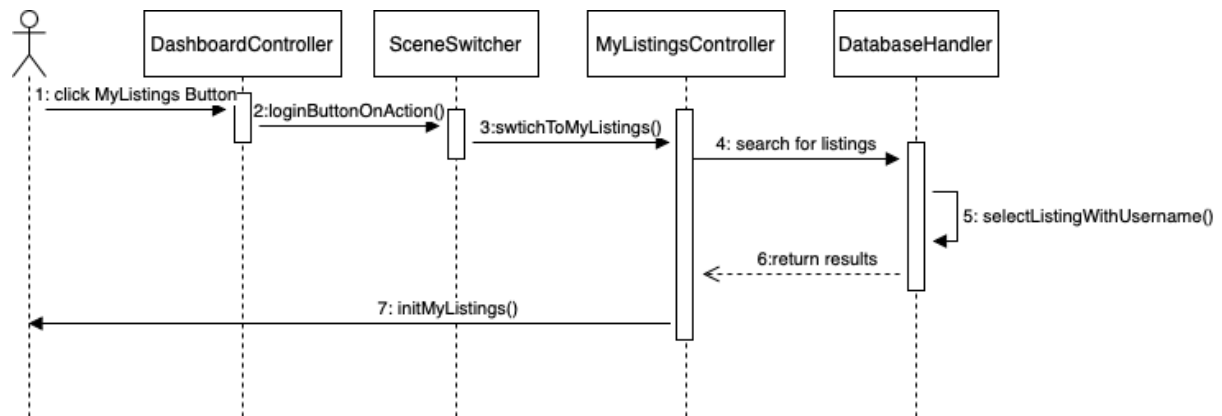


Figure 47: Sequence Diagram-My Listings

The sequence diagram shows the normal process of displaying all listings of a specific user. It also shows the interactions between the different classes in detail.

The process starts in the DashboardController there the user will click on the MyListings button. The user then gets redirected by the SceneSwitcher to the MyListingsController. Here the controller tells the DatabaseHandler to search for all listings the user has. The DatabaseHandler searches the database with the selectListingsWtihUsername method and returns the results. After that the MyListingsController uses the initMyListings method to display the results.

4.1.8 Edit Profile

4.1.8.1 Snowcard

Name	Edit Profile
ID	8
Description	User can edit personal information or delete their account
Trigger	The User click on the “Profile” button on the Dashboard Frame
Actors	User
Pre-Condition	User logged in successfully
Post-Condition	User changes general information and are saved to the database or user deletes the account
Basic flow Description Actions 1 2 3 4	User successfully edits account information User clicks on the “Profile” button on the Dashboard frame The user Changes one or more information (First name, last name, password, city, address) and presses the update button The EditProfil Controller and the DatabaseHandler receive the data and check it for mistakes The DatabaseHandler saves the data and updates the account details
Alternative Flow Description Actions 1 2 3	A The user wants to use a invalid password format The user enters a password that does not meet the applications password requirements EditProfil Controller displays a warning message, that informs the user which requirements are not fulfilled User chooses a different password that meets all requirements
Alternative Flow Description Actions 1 2 3	B Password and password confirmation are not the same User enters a different input in the password and confirm password input fields EditProfil Controller displays a warning, which states, that the inputs are not matching The user corrects the mistake and enters the same password twice
Alternative Flow Description Actions 1 2 3	C User wants to delete the account The user presses the delete button The user decides to delete the account and presses okay as confirmation User gets redirected to the login window

Figure 48: Snowcard-Edit Profil

4.1.8.2 Use Case Diagram

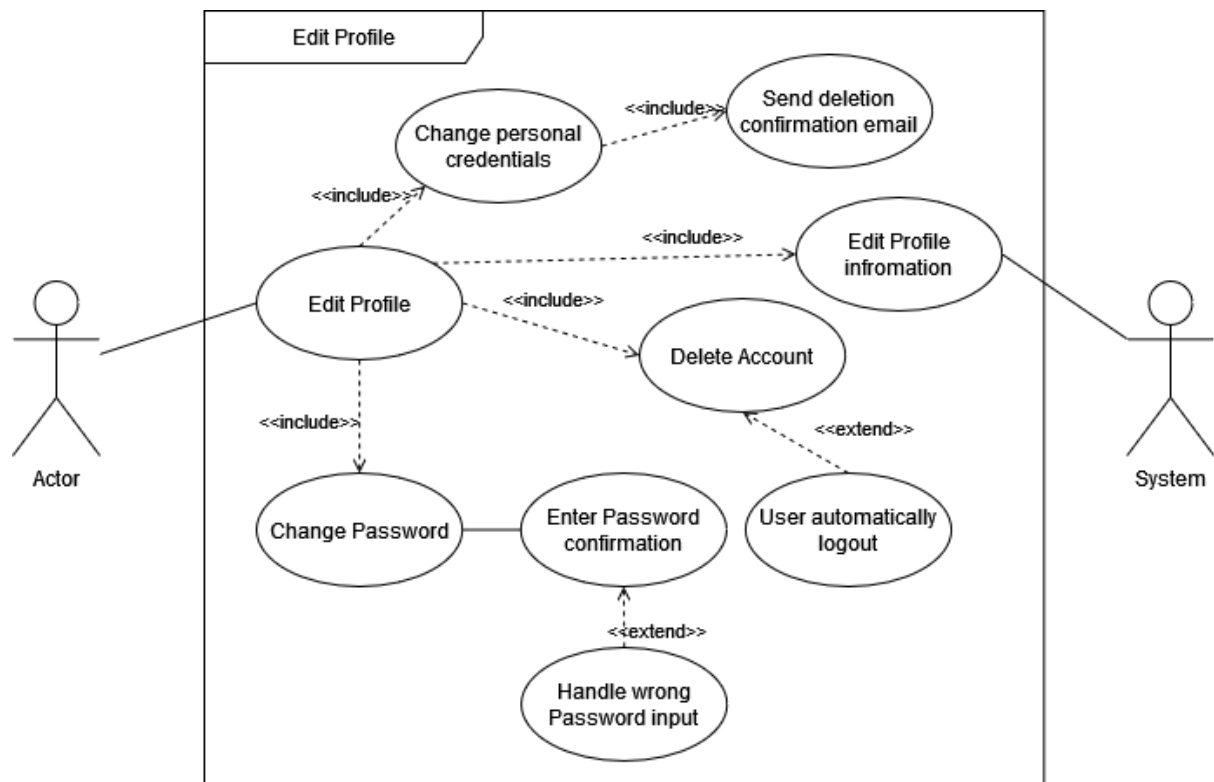


Figure 49: Use Case-Edit Profile

This use case diagram illustrates how the edit of a profile on our platform is supposed to work. As soon as the user opens the profile window, he can change personal information such as first name, last name, city and address as well as the password (twice for verification). The user has also the option to completely delete the account by clicking and confirming the action. After that all user specific entries and listings are deleted and a deletion confirmation email is sent automatically to the user.

4.1.8.3 User Interface

CarSellingPlatform

Edit Profil

User: Max

CSP
Car SellingPlatform

First Name	Last Name
Max	Mustermann
City	Address
Frankfurt am M	Musterstrasse 1
Password	<u>Password Requirements:</u> -One Capital Letter -One digit (1, 2, 3 ...) -One special character (!*+~#...)
Confirm Password	

Update

Back Delete Account

Figure 50: User Interface-Edit Profile

Upon selecting the profile button on the dashboard, the user is navigated to a window displaying all their specific information. This data is automatically fetched and displayed for the user's convenience. If the user decides to make changes and subsequently presses the update button, a message is displayed confirming the successful update, provided no errors are detected and the changes are successfully saved to the database.

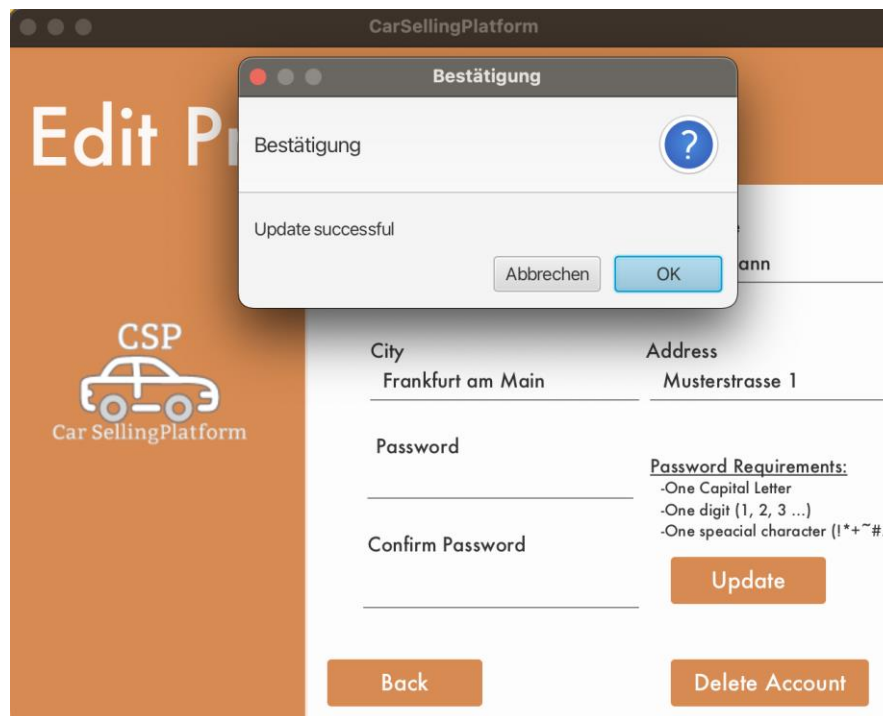


Figure 51: User Interface-Edit Profile update successful

For instance, consider a scenario where the user modifies the city name from “Frankfurt am M” to “Frankfurt am Main” and then presses the update button. The system automatically validates the input, updates the database, and displays a message confirming the successful update.

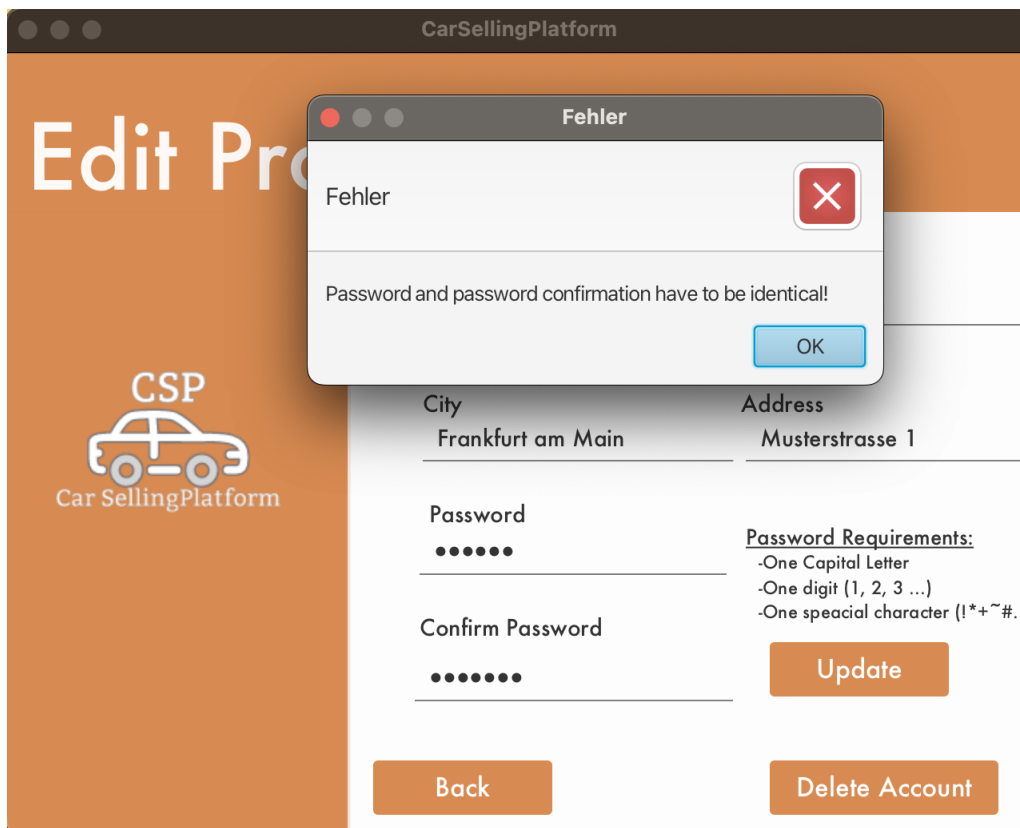


Figure 52: User Interface-Edit Profile Password Error

However, if the user attempts to change the password but enters two different password variations in the password and confirm password input fields, the system detects this discrepancy. Upon clicking the update button, the user is presented with an error message indicating the mismatch.



Figure 53: User Interface-Edit Profile delete User

In another scenario, if the user wishes to delete their account, they can do so by clicking the delete account button. A confirmation message is displayed to ensure the decision was not made inadvertently. Upon confirming the deletion by clicking the OK button, the user is logged out of the application and redirected to the login window. All listings associated with the account are also deleted. Finally, an email is sent to the user confirming the deletion of their account.

4.1.8.4 Sequence Diagram

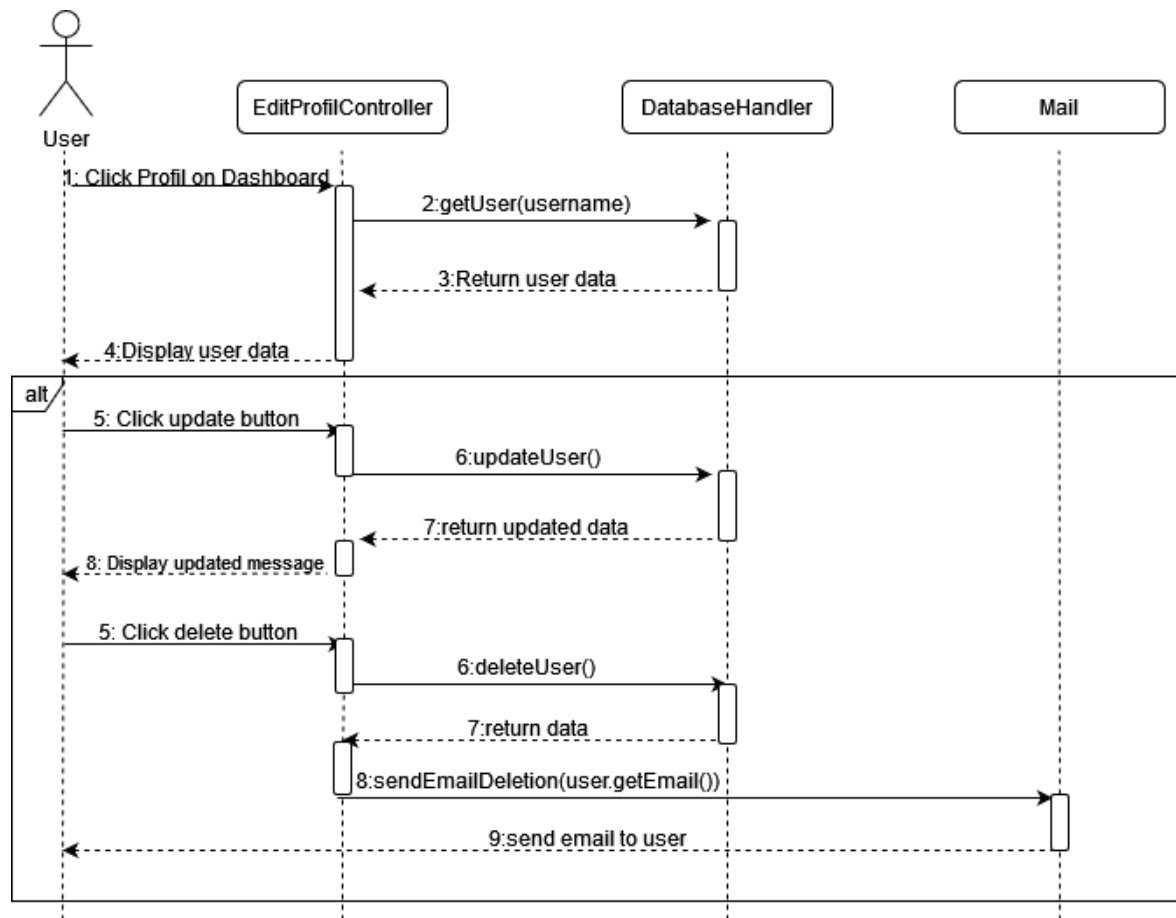


Figure 54: Sequence Diagram-Edit Profile

In the `EditProfilController` class, a series of interactions occur that facilitate the user's ability to view and modify their profile. Initially, when the user selects their profile in the dashboard frame, the `EditProfilController` is triggered. This controller then calls the `getUser(username)` method in the `DatabaseHandler` class to retrieve the user's data. Once the `DatabaseHandler` returns the user data, the `EditProfilController` takes charge of displaying this data in the according text fields.

When the user decides to update their profile and clicks the update button, the `EditProfilController` is once again activated. This time, it calls the `updateUser()` method in the `DatabaseHandler` class to apply the changes made by the user. The `DatabaseHandler` then returns the update status to the `EditProfilController`, which subsequently displays the update status to the user.

If the user wants to delete their profile and clicks the delete button, the `EditProfilController` responds by calling the `deleteUser()` method in the `DatabaseHandler` class. The

DatabaseHandler processes the deletion request and returns the data to the EditProfilController. The EditProfilController then invoking the sendEmailDeletion() which calls the responsible function in the email class and sends an email to communicates this deletion status to the user.

4.1.9 Edit Listings

4.1.9.1 Snowcard

Name	Edit Listing
ID	9
Description	User has the option to edit an existing listing
Trigger	User clicks on the “Edit listing” button on the MyListings frame
Actors	User
Pre-Condition	User is logged in; User wants to edit one of his listings
Post-Condition	The listing information are update and the changes are visible for all other users
Basic flow Description Actions 1 2 3 4 5 6	User successfully edits a listing and changes the picture User clicks “Edit Listing” button User sees all of the information of the chosen listing User adds a picture via the add picture button and chooses a picture from his local harddrive ImageView element gets set to the users chosen picture User clicks on the “Edit Listing” button and the changes are saved in the database EditListing Controller informs the user that the listing got successfully updated and redirects to the MyListing frame
Alternative Flow Description Actions 1 2 3 4 5	A User successfully edits a listing without changing the picture User clicks “Edit Listing” button User sees all of the information of the chosen listing User sees all of the listing attributes and changes their values User clicks on the “Edit Listing” button and the changes are saved in the database EditListing Controller informs the user that the listing got successfully updated and redirects to the MyListing frame
Alternative Flow Description Actions 1 2 3 4 5 6	B User tries to enter other characters than numbers on the price, miles or horsepower input fields User clicks “Edit Listing” button User tries to enter invalid characters for a input field that only accepts numbers as input and presses the button A warning gets displayed to only enter numbers on the affected input fields User enters valid data and follows the rest of the necessary steps to edit the listing User clicks on the “Edit Listing” button and the changes are saved in the database EditListing Controller informs the user that the listing got successfully updated and redirects to the MyListing frame
Alternative Flow Description Actions 1 2 3 4 5	C User leaves at least one input field empty User clicks “Edit Listing” button User forgot to enter a data into a input field and wants to proceed and presses the button A warning gets displayed that tells the user which information is missing User corrects the mistake and presses the “EditListing” button again EditListing Controller informs the user that the listing got successfully updated and redirects to the MyListing frame
Alternative Flow Description Actions 1 2 3 4 5	D User tries to change the cars condition to “new” even though it has pre owners User clicks “Edit Listing” button The User sets the car condition to new and the number of pre owners higher than 0 and clicks the button A warning gets displayed that informs the user of the mistake User corrects the mistake and presses the “EditListing” button again EditListing Controller informs the user that the listing got successfully updated and redirects to the MyListing frame

Figure 55: Snowcard-Edit Listing

4.1.9.2 Use Case Diagram

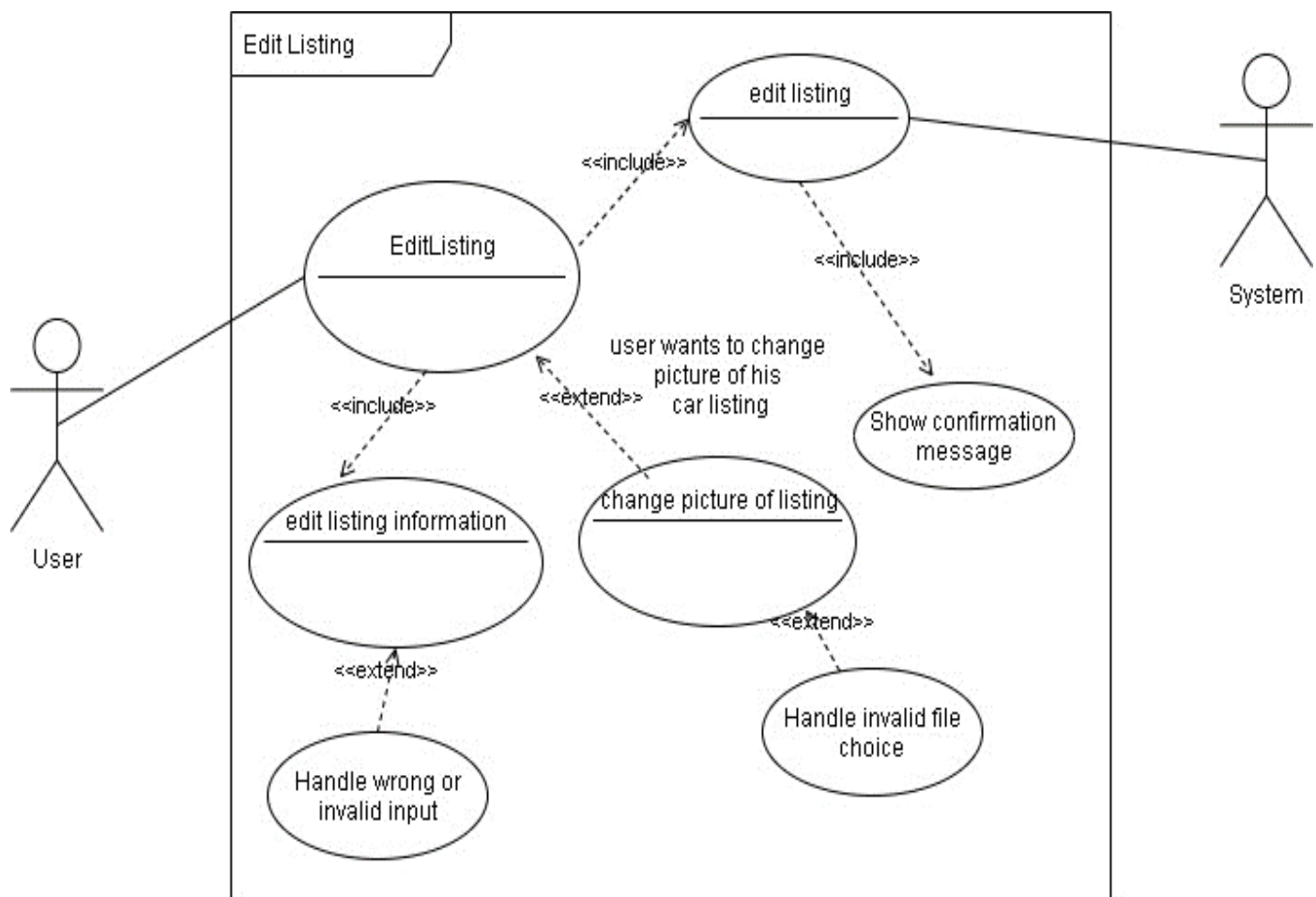



Figure 56: Use Case Diagram-Edit Listing

In this Use case diagram, you can see a simplified version of an interaction between the application and a user that tries to change the data of a listing that already got uploaded. This requirement allows the user who uploaded a listing to change any attribute of his listing, such as price, condition etc. The input handling and conditions are the same as in the “create Listing” requirement.

4.1.9.3 User Interface

Edit your car Listing



Change Picture

<div>Listing Name</div> <div>Ferrari Roma</div>	<div>Price</div> <div>285000.0 €</div>	<div>Condition</div> <div>Used</div>
<div>Brand</div> <div>Ferrari</div>	<div>Model</div> <div>Roma</div>	<div>Body type</div> <div>Coupe</div>
<div>Colour</div> <div>Silver</div>	<div>Number of doors</div> <div>2/3</div>	<div>Number of seats</div> <div>2</div>
<div>Interior Material</div> <div>full leather</div>	<div>Air Bags</div> <div>Front- & Side-Airba..</div>	<div>Air Conditioning</div> <div>2-zone automat..</div>
<div>Registration Date</div> <div>19.05.2023</div>	<div>Last general inspection</div> <div>20.05.2023</div>	<div>Number of Pre-Owners</div> <div>1</div>
<div>Emission Class</div> <div>Euro6</div>	<div>Environmental Badge</div> <div>Green</div>	<div>Consumption (l per 100km)</div> <div>9.0</div>
<div>Engine Type</div> <div>V8</div>	<div>Performance (Horse Power)</div> <div>800</div>	<div>Fuel type</div> <div>Petrol</div>
<div>Cylinder Capacity</div> <div>3000</div>		<div>Gearbox</div> <div>Automatic</div>

Description

Ferrari fast

Change Listing

Go Back

Figure 57: User Interface-Edit Listing

Here the JavaFX scene that is presented to the user, when wanting to change an already existing listing can be seen. The scene itself looks nearly identical to the “create Listing”-frame but loads all of the existing listing’s attributes beforehand and presents them in an editable manner to the user. The data gets loaded on a way, that it looks like the user just created the listing, which makes it more user friendly and intuitive for the user to change any attribute values, the rest of the logic and usage is like the “createListingController” functionalities.

4.1.9.4 Activity Diagram

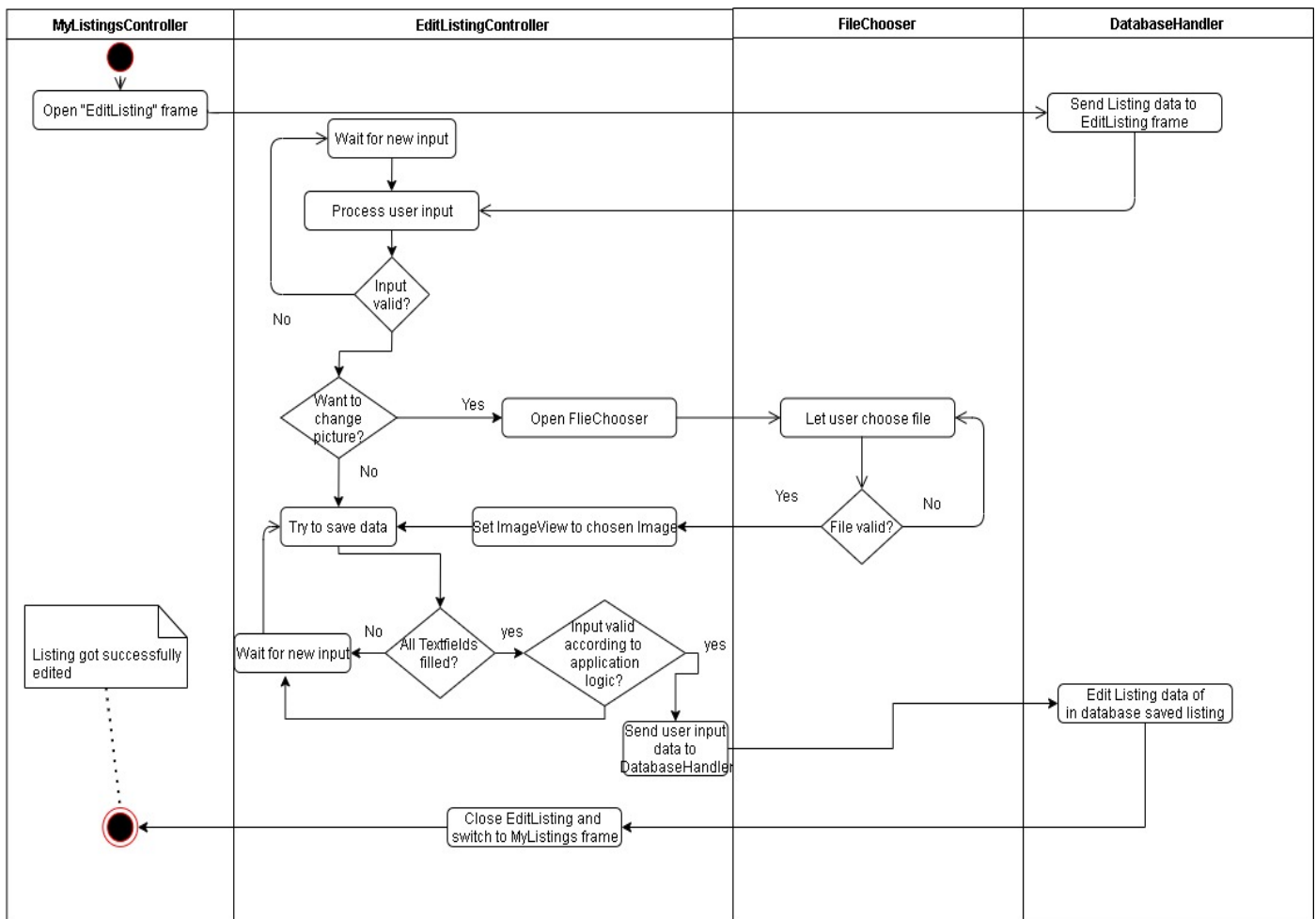


Figure 58: Activity Diagram-Edit Listing

This activity diagram visualizes the flow of activities, the requirement of editing an existing listing brings with it. The editing itself starts with the invocation of a “EditListing”-Frame, which is accomplished by using any “Edit Listing” button on one of the user’s on the “MyListings” presented listings. When the scene opens, it loads all of the listing’s data in a “createListing” similar frame, where the can then change any of the listings information, if the user tries to change any of the values in a way, that the application should not accept (like letters for numerical values),the user gets directly interrupted, and told by a warning message, that the input he tried to enter is invalid, and which kind of input is accepted in this case, so the user can then continue and correct his mistake. The same goes for any other different try of application logic violations, because the “EditListing” frame uses the same input handling and processing, like the “createListingController” class.

4.1.9.5 Sequence Diagram

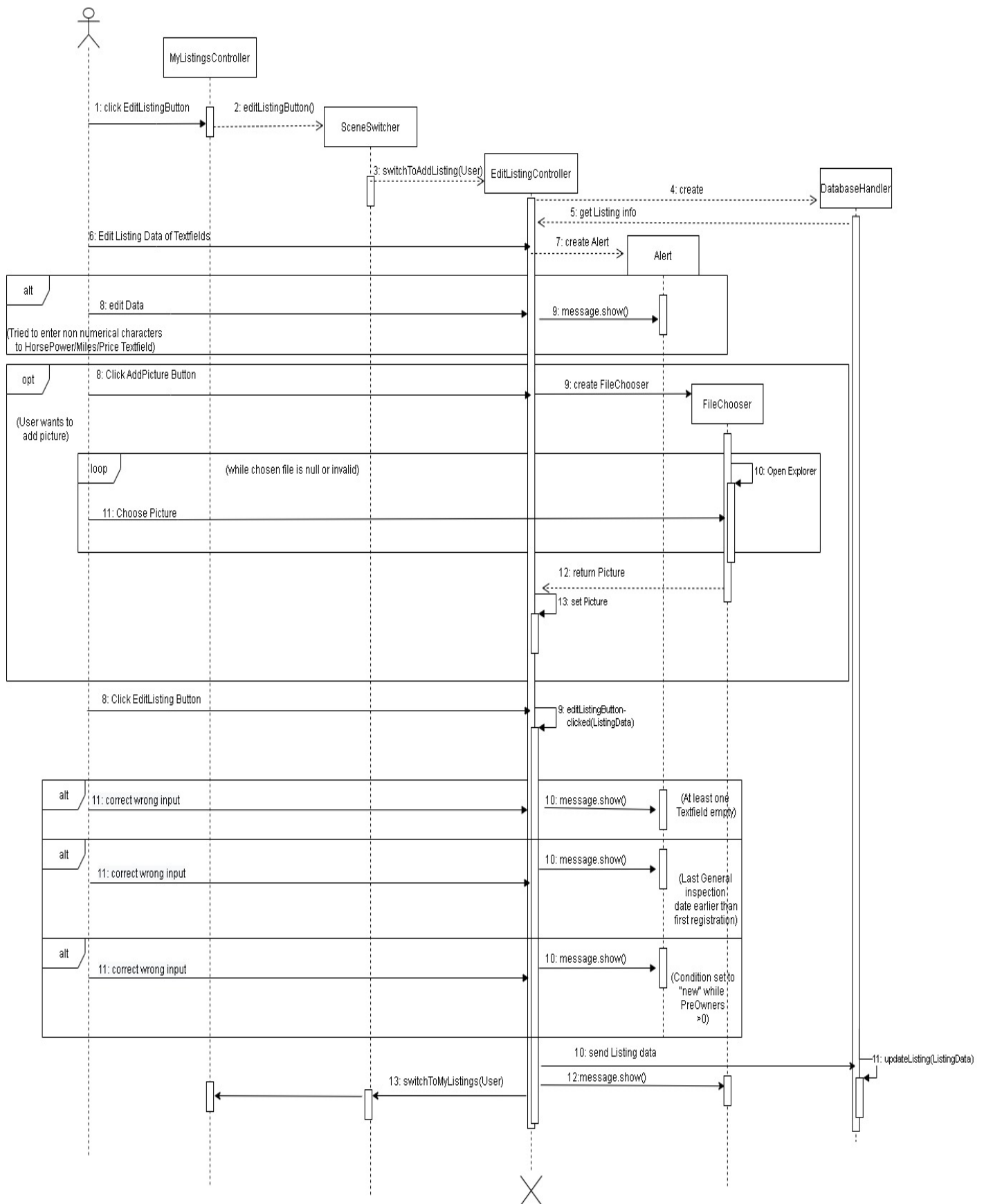


Figure 59: Sequence Diagram-Edit Listing

Here you can see the sequence diagram of this requirement, which visualizes the interaction/sequence flow of the user and the application's different classes, when a user edits an existing listing. First the user has to use an "Edit Listing"-button, which is only available on the "MyListing"-frame, from there on, an instance of the SceneSwitcher class gets created, which changes the scene, the user is presented, from the "MyListings"-frame to the "EditListing"-frame (here the SceneSwitcher uses the switchToEditListing method, which takes the user's information and the affected listing's ID as parameter to be able to load the correct required data afterwards). Here a newly created instance of the "EditListingController" creates an instance of the database handler, which fulfills the role of a database interface and grants access to the application's database, which allows the EditListingController to load all attribute values of the correct listing, on the "EditListing"-frame. After that is done, the EditListingController creates an instance of the JavaFX standard class "Alert", which gives the application the possibility to show message dialogs and warnings to the user in certain conditions. The instance of the "Alert" class gets only used in case of an application logic violation by the user, while changing a listing's data in this case. The user also gets presented the option to voluntarily change his listing's picture, where the "FileChooser"-class gets used to offer the user the possibility of choosing a valid file for the new picture, the user can choose between files, until he tries using a valid one (the user can not choose an invalid file type, which would violate the application's foreseen kind of usage). After the user chose a new file (or may not), he can try to finish his editing by using the "Edit Listing"-button which invokes the EditListingButtonClicked() method, which takes all of the listing's entered attributes as parameters from this scene's textfields and choice boxes, here the data gets processed, and if there are any mistakes, the dataset will not be saved, and the "EditListing"-frame will remain open until the user fixes his mistakes and tries updating the listing again, or until he may use the available "back" button which would redirect the user back to the "MyListings"-frame, and would not save any of the users changes. But if all of the user's changes are valid, the data gets sent to the DatabaseHandler, which then uses its "updateListing()" method to apply the user's editing choices on the saved listing dataset. After that is done, an alert message pops up that tells the user that his listing got successfully updated. After that, the "EditListing"-frame closes itself and uses the SceneSwitcher to redirect the user back to the "MyListings"-frame.

4.2 Communication via Mail


 Mail
<ul style="list-style-type: none"> □ username: String □ password: String □ props: Properties
<ul style="list-style-type: none"> ● prepareSession(): Session ● sendEmailRegistration(recipientEmail: String): void ● sendEmailDeletion(recipientEmail: String): void ● sendOnetimeCode(recipientEmail: String, code: String): void ● sendNewPasswordConfirmation(recipientEmail: String): void ● sendMessage(recipientEmail: String, senderEmail: String, senderUsername: String, listing: String, text: String): void

Figure 60: Mail-Class

The Mail class is responsible for sending various types of emails in the context of a car selling platform. It uses the JavaMail API for sending emails over SMTP.

The class has several static methods for sending different types of emails:

1. **sendEmailRegistration:** This method is used to send an email to a recipient email address when a user registers on the car selling platform. The email contains a welcome message and information about the platform.
2. **sendEmailDeletion:** This method is used to send an email to a recipient email address when a user requests to delete their account. The email contains a goodbye message, appreciation for their participation, and information about the consequences of deleting the account.
3. **sendOnetimeCode:** This method is used to send an email to a recipient email address containing a one-time code for resetting the password. The email instructs the user to enter the provided code and warns them not to share it with anyone.
4. **sendNewPasswordConfirmation:** This method is used to send an email to a recipient email address to confirm that their password has been successfully updated.
5. **sendMessage:** This method is used to let a user write another user about a listing. The email includes the sender's username, the listing they wrote about, and the message content. It also provides a link to reply to the sender via email.

The class uses the Config class to retrieve the email username and password. It sets up the necessary properties for the mail session, such as enabling authentication and specifying the SMTP server details (in this case, "mail.gmx.net" with port 587).

5. Sources

Getting Started with JavaFX: About This Tutorial | JavaFX 2 Tutorials and Documentation.

(2013, August 30). https://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm

java.sql (Java Platform SE 8). (2023, April 5).

<https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html>

javax.mail (Java(TM) EE 7 Specification APIs). (2015, June 1).

<https://docs.oracle.com/javaee/7/api/javax/mail/package-summary.html>

Quickstart. (n.d.). Clever Cloud Documentation. <https://www.clever-cloud.com/doc/getting-started/quickstart/>

Villan, M. A. (2018, September 23). *JavaFX Scene Builder Tutorial for Beginners*. Genuine Coder. <https://genuinecoder.com/javafx-scene-builder-tutorial-for-beginners/>