

Software Requirements Specification

- [1. Introduction](#)
 - [1.1. Overall Description](#)
 - [1.1.1. Physical system](#)
 - [1.1.1.1. Wiring model](#)
 - [1.1.1.2. Glossary](#)
 - [1.1.2. Software purpose](#)
 - [1.1.3. System flow](#)
 - [1.1.4. Functions](#)
 - [1.1.4.1. Existing functions](#)
 - [1.1.4.2. New functions](#)
 - [1.2. User Classes and Characteristics](#)
 - [1.3. Operating Environment](#)
 - [1.3.1. Minimum hardware requirements](#)
 - [1.3.2. Operating systems and versions](#)
 - [1.3.2.1. Supported operating systems](#)
 - [1.3.2.2. Alternative operating systems](#)
 - [1.4. Design and Implementation Constraints](#)
 - [1.5. Product Functions](#)
- [2. Domain Model](#)
- [3. Use case Descriptions](#)
 - [3.1. Add car to user](#)
 - [3.2. Add car to system](#)
 - [3.3. Register account](#)
 - [3.4. Login](#)
 - [3.5. Manage personal data](#)
 - [3.6. Charge car](#)
 - [3.7. Update user](#)
 - [3.8. Fill in car's battery percentage](#)
 - [3.9. Add measurement](#)
 - [3.10. Check charging status](#)
 - [3.11. Verify users](#)
 - [3.12. Verify cars](#)
 - [3.13. Check charging socket graph data](#)
- [4. Other functional requirements](#)
- [5. Non-functional Requirements](#)
 - [5.1. Efficiency](#)
 - [5.2. Security](#)
 - [5.3. Reliability](#)
 - [5.4. Usability](#)
 - [5.5. Maintainability](#)
 - [5.6. Portability](#)
- [6. User interface sketches](#)
 - [6.1. Base design](#)

- [6.2. Homepage](#)
- [6.3. About](#)
- [6.4. Login](#)
- [6.5. Register](#)
- [6.6. My cars](#)
- [6.7. Add a car to user](#)
- [6.8. Add a new car](#)
- [6.9. Battery fill in](#)
- [6.10. Verify users](#)
- [6.11. Verify cars](#)
- [6.12. Account info](#)
- [6.13. Admin page send email](#)
- [7. Resources](#)

1. 1. Introduction

This document describes the functions and requirements of the system to be developed. The purpose of this document is to give the project group and stakeholders a clear picture of the functional aspects and behaviour of the system to be developed. With the help of this document, you can understand the application at a glance without any technical knowledge.

1.1. 1.1. Overall Description

A member (Trung Nuygen) at the HAN Control System Research Group (CSRG), has a project that runs in a series across multiple project teams. The project is about a system called Sevci. Sevci is used to registers users that wish to charge their electric car and register the power used to charge the car.

The most innovative feature of the system is the method of charging. This is currently being done through grid power. However, the HAN also has solar panels that can charge the cars. This solar power could go directly to the car or to the storage where the generated power could be used later on. By placing smart meters on the network, it is possible to keep track of exactly how much power is being sent.

1.1.1. 1.1.1. Physical system

To understand how the IoT works with solar panels, power storage and power grid work, a wiring model as been made to show how they communicate with each other. Below is also a glossary stating the different components and what they do.

1.1.1.1. 1.1.1.1. Wiring model

The charging pole communicates with a smart power meter. The power meter registers the in coming power. The solar panels sends power to the power storage or charging pole. The power storage can also send power to the charging pole. If there is not much power coming from the

solar panel or power storage the rest of the power will be taken directly from the power grid. For more information about each component see the next glossary.

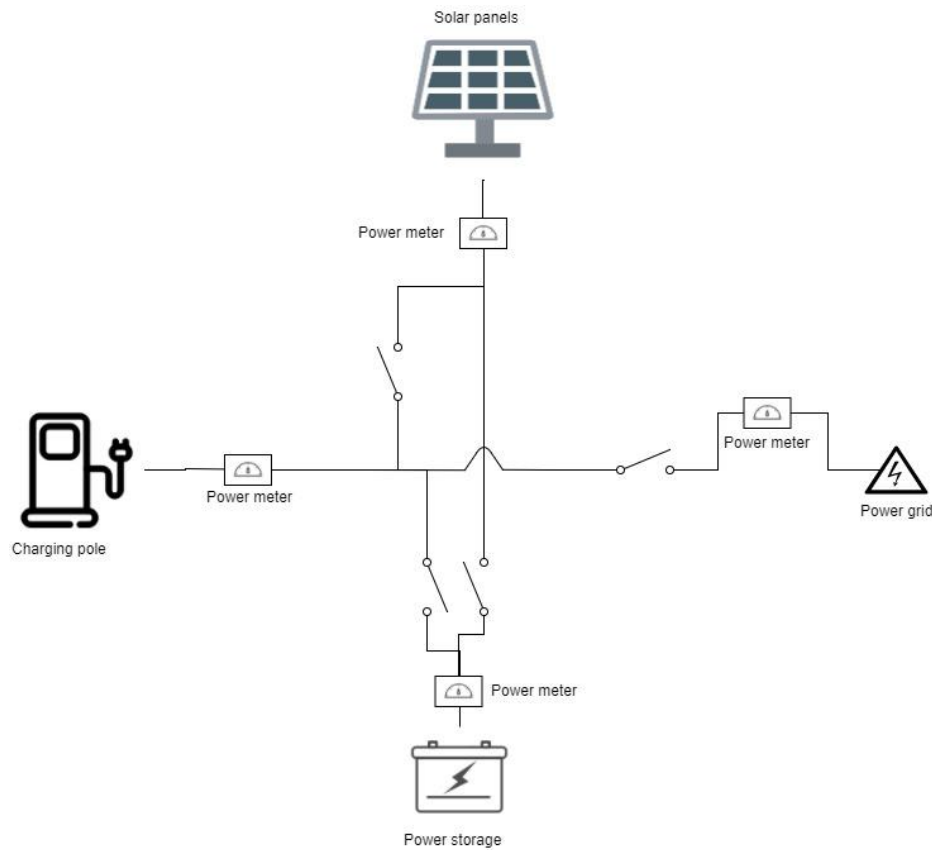


Figure 1: Wiring Model

1.1.1.2. 1.1.1.2. Glossary

Concept	Description
Charging pole	Cars can connect to one of the two sockets present on the charging pole. The charging pole then powers the car using electricity. Using data from the power meters, the charging pole can decide where it wants to get its power from. It can directly come from the solar panels, from the power storage or it can get the power of the standard electrical power grid.
Solar panel	The solar panel generates electricity through the sun which can be stored in the power storage or can be used by the charging pole.
Power storage	When no car is connected, solar energy can be stored in this storage for later use.

Electrical grid	The electrical grid represents the normal electricity grid. When no power can come from the power storage or from the solar panels, electricity will flow from this grid. In another scenario, where there is electricity available at the solar panels, the administrator can still decide to use the electricity from the electrical grid.
Power meter	Power meters are connected in several places as seen in the figure above. They measure the amount of current that flows through that point so the charging pole / administrator knows where electricity is stored. These power meters will store their measures in the database.

Table 1: Glossary of wiring modal

1.1.2. 1.1.2. Software purpose

The purpose of this system is for users to be able to charge their car and see if a socket is free for their car to attach to.

Purpose of the software:

- To charge cars
- To sign in users
- To see how far a car is charged
- To notify a user that his car has to be moved
- Register data from the solar panel power meter

1.1.3. 1.1.3. System flow

If someone wants to charge their car they are required to register an account in Sevci. After they fill in the register form the administrator will accept or deny their account. Once accepted they can add a car to their account. If the car they want to charge is in the list it will be added to the account. If the car is not in the list the user can request adding a new car to the list and use that, this requires an administrator to accept or deny the car request. Now the car can be selected and the user will be able to say how full their car battery is at that moment. They are now required to check in at that charging pole and connect the car. Once the car is fully charged the user will receive an email asking the user to remove their car.

1.1.4. 1.1.4. Functions

1.1.4.1. 1.1.4.1. Existing functions

The basic functionalities are:

- Login as a user or administrator
- Verify cars that are not yet in the car list
- Verify users that would like to use the Sevci system
- A user can edit their personal information

- Email function is built but not completely working

1.1.4.2. 1.1.4.2. New functions

Seeing as the basic functionalities already exist, the functions that will be added are:

- Solar panel data will be saved to a database in Azure
- A connection from the Java REST API to the power meter reader of the solar panel
- Administrators can send an email to a user that is currently charging their car
- Display power levels from a socket in a graph
- The web application is mobile compatible (view-able in a mobile browser)

Next to these new functions, there will be a lot of refactor work for system maintainability.

More information about the problem and the goal can be found in the action plan under the [Goal, task, and results for the company and school](#) chapter.

1.2. 1.2. User Classes and Characteristics

In this paragraph, the actors within the system are included as is a description of who it represents.

Actor	Description
Guest	A guest is a visitor of the site that is not logged in. The moment a guest is logged in, it becomes a user. The precondition of being a guest is that he is not logged in.
User	A user is a registered user in the system and is logged in to the system. The preconditions of being a user is that he is verified by an administrator and is logged in.
Administrator	A user that is registered, logged in and marked as an administrator in the system. The administrator is an extend of the user. This means every action the user can preform, the administrator can also preform. As the actor name already states an administrator can do administrative tasks that a user can't do. The preconditions of being a administrator are the same as the user in extend that he needs to be verified as administrator.
Charging station	This is a system actor and represents the charging station's IoT device.
Smart meter	This is a system actor that represents the meters that are put on the electrical grid to measure the power going through the grid.

Table 2: User Classes and Characteristics

1.3. 1.3. Operating Environment

1.3.1. 1.3.1. Minimum hardware requirements

Since the back-end application will be developed in Java, the minimum requirements for Java must be taken into account. With these requirements, sufficient resources are available to run the supported operating systems including the back-end application. For the front-end, NodeJS will be needed. Besides the applications, the application also uses a small MySQL 8.0 database.

For JRE 8 (Oracle, 2016):

- 128 MB RAM
- 124 MB available disk space
- 2 MB for the Java Update
- 1 GHz processor

For NodeJS 11:

- 64 bit operating system
- 128 MB RAM
- 124 MB available disk space
- 1 GHz processor

1.3.2. 1.3.2. Operating systems and versions

1.3.2.1. 1.3.2.1. Supported operating systems

The application is developed and tested in Windows 10 1803. This is because the development team all works with Windows machines.

1.3.2.2. 1.3.2.2. Alternative operating systems

The application can be further installed on the operating systems that can run at least JRE version 8 and NodeJS 11. However, we do not guarantee the correct operation of these on these systems, because these operating systems will not be tested.

1.4. 1.4. Design and Implementation Constraints

In this paragraph, the issues that will limit the options available to the developers are described. These include hardware, technologies, tools, and databases to be used, programming language required and communications protocols.

Code	Constraint
DIC1	Java is used as a programming language for the back-end application.
DIC2	Spring is used as a framework for running the back-end application.
DIC3	Angular is used as a framework for running the front-end application.
DIC4	Communication protocols have been established between the charging station and the application that must be used and not altered.
DIC5	MySQL is used as a database for data storage.
DIC6	Azure is used as a hosting server and the application must be able to run on this server.
DIC7	JUnit 4, Mockito and PowerMock are used for testing the back-end application.
DIC8	Karma and Jasmine are used for testing the front-end application.

Table 3: Design and Implementation Constraints

1.5. 1.5. Product Functions

In this paragraph, the major functions the product performs or the user performs on the product are found. This can be seen in the use case diagram below. In chapter [Use case Descriptions](#) more detail can be found about these functionalities.

1.5.1.1.1.1. 1.5.1.1.1.1. Use case diagram

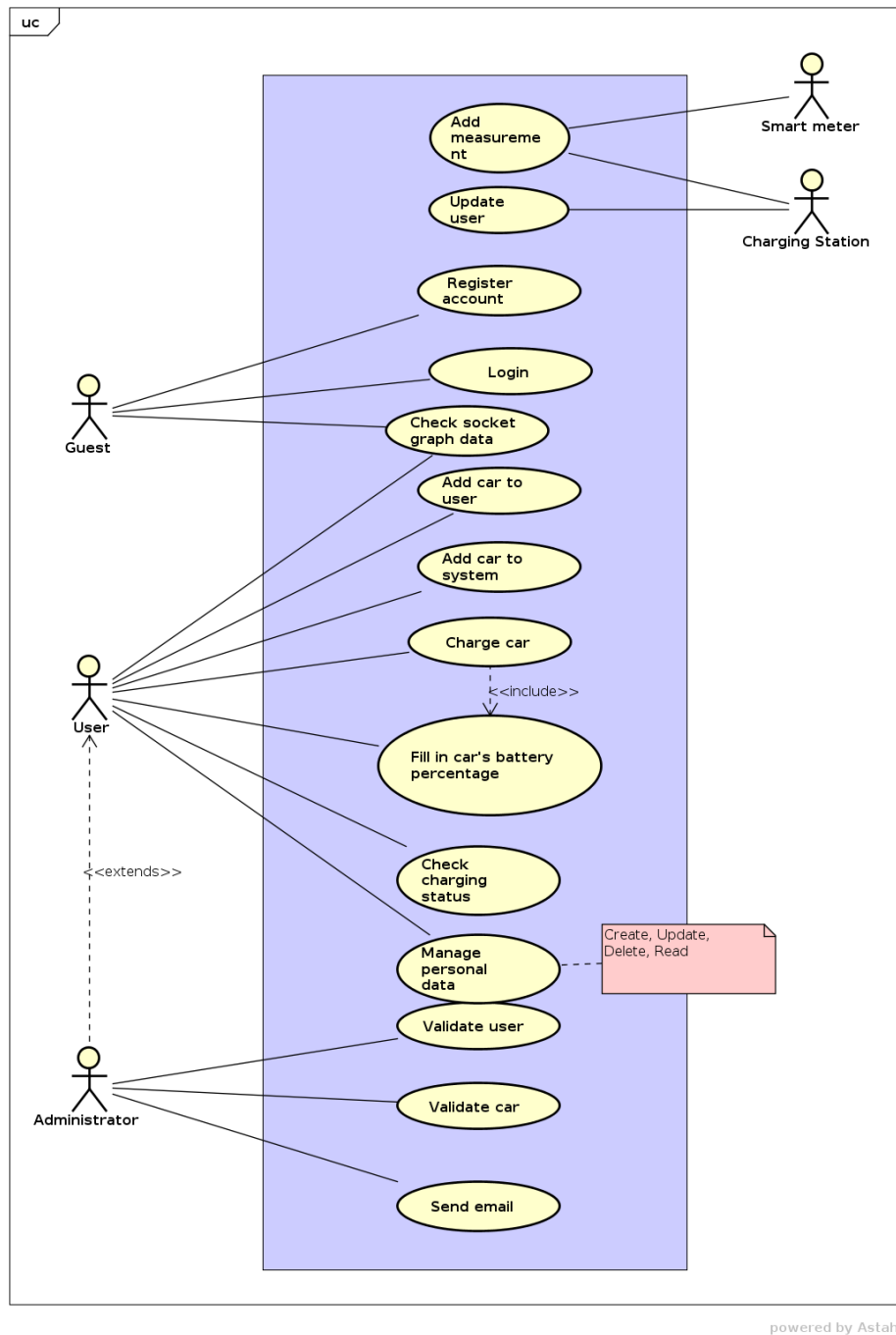


Figure 2: Use case diagram

Below are all use cases listed with there brief description. By selecting a use case, the page with the full use case will be opened.

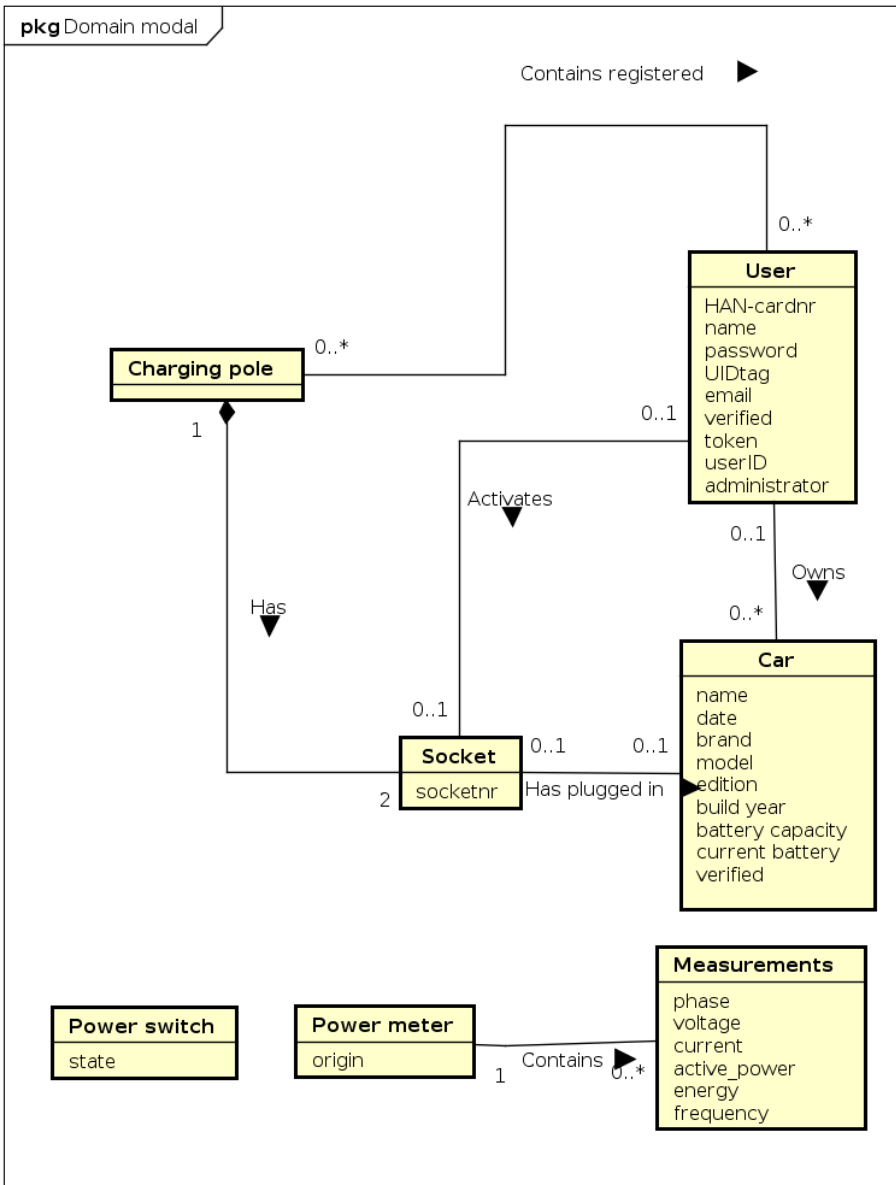
Use case	Brief description
Add car to system	The user can add a new vehicle to the system.

Use case	Brief description
Add car to user	The user can add a car to their list of cars that they own.
Add measurement	A charging station adds measurements to the database while a user is charging his car. These can be used to calculate the remaining charging time.
Charge car	The user can start charging his car at the charging station.
Check charging socket graph data	A Guest or User can check how much the car has been charged.
Check charging status	A user can check the charging status of the car being charged
Fill in car's battery percentage	The user can fill in the car's battery percentage of their car.
Login	The guest can log in to the system
Manage personal data	The user can create, read, update or delete their personal information.
Register account	A guest can create an account
Update user	The charging station updates the user's start / stop time that is currently charging and sends a signal to the other charging stations to update them.
Verify cars	The administrator can verify new cars that have been added by an user.
Verify users	The administrator verifies guests that request an account. He can either decline or accept these requests.

2. 2. Domain Model

In the figure below you can find the domain model. In the glossary all items are included to explain them.

2.1.1.1.1.1. 2.1.1.1.1.1. Domain model



powered by Astah

Figure 3: Domain model

Glossary

Concept	Description
User	A user of the system who can be an administrator and has more rights than a standard user.

Car	A car in the database. Does not have to be linked to a user. This can happen at any given moment.
Socket	A socket is built into a charging pole. They contain a number and, every pole has two.
Charging pole	The charging pole has two sockets. It also contains a list of registered users that can use the charging sockets.
Power meter	The power meter registers the power coming from his origin. These gets saved in measurements.
Power switch	It opens or closes paths for energy to travel through. The variable "state" represents if this is open or closed.
Measurements	Contains the measurements of power meters.

Table 4: Glossary of domain model

3. 3. Use case Descriptions

In this section, each use case is described in detail, optionally accompanied by a system sequence diagram and operation contracts. As explained in [the introduction](#), the precondition for a user is always that the he is logged in. If the primary actor is an administrator, he also needs to be verified as administrator. An administrator can do the same things as a user but extends him in administrative functionalities.

3.1. 3.1. Add car to user

Use case	Add car to user
Primary actor	User
Stakeholders and Interests	None
Brief description	The user can add a car to their list of cars that they own.
Preconditions	None

Postconditions (Success Guarantee)	Car has been successfully added to a user's car list.
Main success scenario (Basic flow)	
1. User opens the 'add a car to user' page.	
	2.The system shows the page add a car to user .
3. The user chooses a car in the list of verified cars.	
4. The user clicks accept button	
	4. The system saves the selected car to the user's list of cars and shows a message that it has been added to their list. [End of use case]

Table 5: Use case - Add car to user

3.2. 3.2. Add car to system

Use case	Add car to system
Primary actor	User
Stakeholders and Interests	None
Brief description	The user can add a new car to the system.
Preconditions	None
Postconditions (Success Guarantee)	<ul style="list-style-type: none"> A new car is added to the system as a non-verified car.
Main success scenario (Basic flow)	
1. User opens the 'Add a new car' page to add a new car.	
	2. The system shows the page add a new car The page shows a header component with a hamburger menu. The page shows a container what contains an input form with a Brand, Model, Edition, Build year and Battery capacity input form according to the design.
3. User fills in the form	

	4. The system saves the new car and shows a message that it has been sent to be verified. [End of use case]
Extensions (Alternative flow)	
A: An incorrect input data	
3A. The user has filled in the form incorrectly.	
	4A. The System rejects the request and sends an error message back to the user from the server. [Proceed to step 3]

Table 6: Use case - Add car to system

3.3. 3.3. Register account

Use case	Register account
Primary actor	Guest
Stakeholders and Interests	None
Brief description	A guest can create an account
Preconditions	<ul style="list-style-type: none"> ▪ The guest has no account ▪ The guest is an employee of the HAN

Postconditions (Success Guarantee)	<ul style="list-style-type: none"> A new user will be saved in the database as a non-verified user
Main success scenario (Basic flow)	
1. The guest clicks on Register	
	2. The system loads the register page
3. The user fills in the registration form	
	4. The system receives the data from the registration form and saves it into the database and shows an okay message [End of use case]
Extensions (Alternative flow)	
A: An Account already exists	
3A. The user fills in the registration form with data (HAN ID, email address) that already exists in the database	
	4A. The system rejects the request because the user already exists and shows an error [Proceed to step 3]
B: Invalid data	
3B. The user fills in the registration form with incorrect data (incomplete data, special symbols and numbers or letters in the wrong places)	
	4B. The system rejects the request because of the invalid data and shows an error [Proceed to step 3]

Table 7: Use case - Register account

3.4. 3.4. Login

Use case	Login
Primary actor	Guest

Stakeholders and Interests	User
Brief description	The guest can log in to the system
Preconditions	<ul style="list-style-type: none"> ▪ The guest is already registered as a verified user
Post conditions (Success Guarantee)	<ul style="list-style-type: none"> ▪ The guest is logged in ▪ The guest is now a user/admin
Main success scenario (Basic flow)	
1. Guest opens the login page	
	2. The system shows the login page .
3. Guest fills in their HAN ID and their password information.	
	4. System validates the HAN ID and the password.
	5. System creates new user token.
	6. The system gives the guest access to the system.
	[End of use case]
Extensions (Alternative flow)	
A. Access denied	
3A. Guest fills in the wrong password or HAN ID	
	[System validates the HAN ID and the password.= false] 4A. System grants no access for the user and shows an error message. [End of use case]

Table 8: Use case - Login

3.5. 3.5. Manage personal data

Use case	Manage personal data
Primary actor	user
Stakeholders and Interests	None
Brief description	The user can create, read, update or delete their personal information.
Preconditions	The user is verified
Postconditions (Success Guarantee)	<ul style="list-style-type: none"> ▪ The user can view their personal information. ▪ The system has updated the data
Main success scenario (Basic flow)	
1. The user opens the page to view their personal information	
	2. The system shows the page account info. with the user's data.
3. The user fills in the form	
4. The user submits the new information	
	5. The system registers the information [End of use case]
Extensions (Alternative flow)	
A: An incorrect data input	
3A. The user fills in incorrect data.	
	5A. The System rejects the request and sends an error message back. [Proceed to step 3]

Table 9: Use case - Manage personal data

3.6. 3.6. Charge car

Use case	Charge car
Primary actor	User

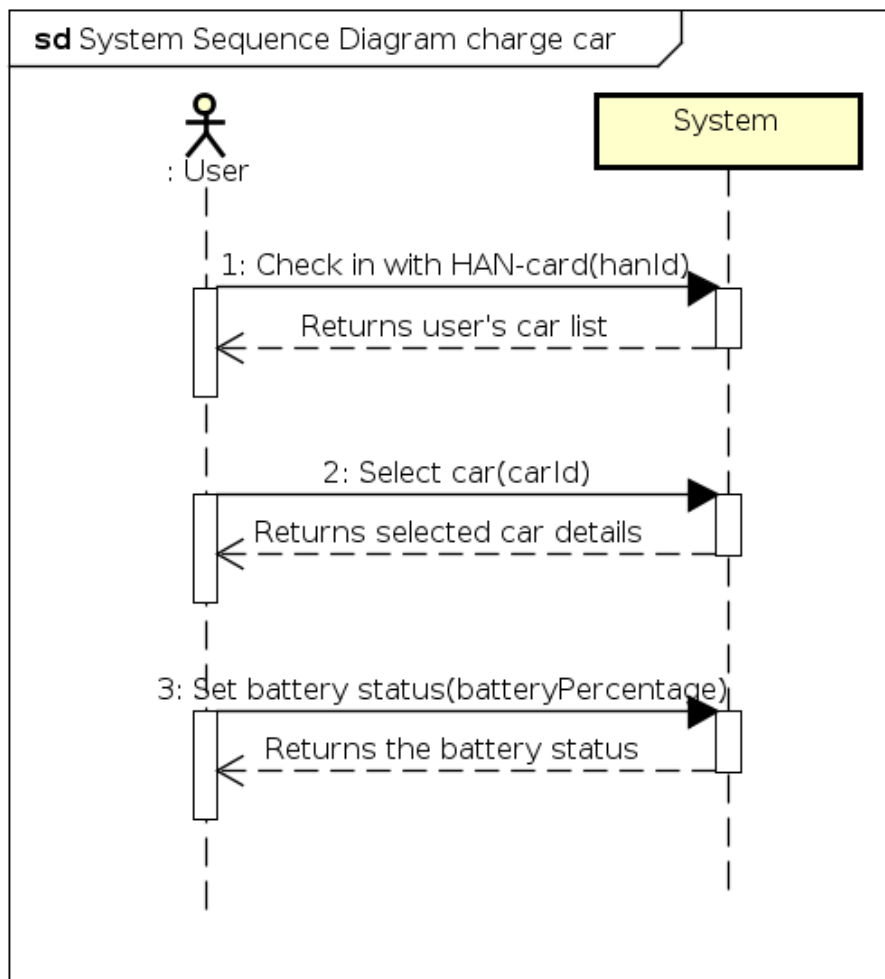
Stakeholders and Interests	Charging station
Brief description	The user can start charging his car at the charging station.
Preconditions	<ul style="list-style-type: none"> ▪ The user has at least one registered car.
Postconditions (Success Guarantee)	<ul style="list-style-type: none"> ▪ The status of the charging pole updated. ▪ The amount of energy is logged. ▪ The user and car of the energy pole are logged.
Main success scenario (Basic flow)	
1. The user checks in with his HAN-card.	
	2. The system checks the user in.
	3. The system executes [use case update user]
4. The user executes [use case fill in car's battery percentage]	
[End of use case]	
Extensions (Alternative flow)	
A: User has no valid Han card	
1A. The user scans their HAN-card.	
	2A. The system rejects the sign in process and throws an error to the user. The error is a response from the rest API.
	[Proceed to step 1]

Table 10: Use case - Charge car

System Sequence Diagram

In this use case, the user needs to check in with his HAN-card before any further actions can be taken. When the user is accepted, he can select his car on the [battery-fill-in page](#). After the user has selected his car, he can set the battery status. This status represents his battery percentage and is used by the system to calculate when the car is done charging. The user then will receive an email that his car is full so he can go and remove his car from the charging pole.

3.6.1.1.1.1. System Sequence Diagram charge car



powered by Astah

Figure 4: System Sequence Diagram Charge Car

3.7. 3.7. Update user

Use case	Update user
Primary actor	Charging station

Stakeholders and Interests	User
Brief description	The charging station updates the user's start / stop time that is currently charging and sends a signal to the other charging stations to update them.
Preconditions	<ul style="list-style-type: none"> A user is charging his car at a charging station socket.
Postconditions (Success Guarantee)	<ul style="list-style-type: none"> The date the user started charging is updated in the database. The other charging stations have their user list updated.
Main success scenario (Basic flow)	
1. Charging station request user with their primary (first found) car from the system.	
	2. System returns list of users with their primary cars.
3. Charging station sends a user update for the start time of the charging session.	
	4. System updates start time of charging in the database.
	5. System sends signal to other Charging stations that a user got updated.

Table 11: Use case - Update user

3.8. 3.8. Fill in car's battery percentage

Use case	Fill in car's battery percentage
Primary actor	User or Administrator
Stakeholders and Interests	None
Brief description	The user can fill in the car's battery percentage of their car.
Preconditions	<ul style="list-style-type: none"> The user has at least one registered car.

Post conditions (Success Guarantee)	<ul style="list-style-type: none"> The battery percentage of the car is filled in and updated in the database.
Main success scenario (Basic flow)	
1. The user opens the page 'Battery fill'	
	2. The system shows the page "Battery fill in" .
	3. The system retrieves a list of cars of the user and shows it to the user
5. The user selects a car from their car list	
	6. The system retrieves and shows the data of the car
7. The user fills in the battery percentage	
	8. The system updates the car information
	9. The system displays the new percentage [End of use case]
Extensions (Alternative flow)	
A: An incorrect input data	
[input < 0 or input > 100] 8A. The user fills in a wrong input	
	9A. The system rejects the request and shows an error message to the user. [Proceed to step 7]
B: User has no registered cars	
	3. The user gets an error message that there is no car found. [End of use case]

Table 12: Use case - Fill in car's battery percentage

3.9. 3.9. Add measurement

Use case	Add measurement
Primary actor	Charging station
Stakeholders and Interests	None
Brief description	A charging station adds measurements to the database while a user is charging his car. These can be used to calculate the remaining charging time.
Preconditions	<ul style="list-style-type: none"> A user is charging his car at the charging station.
Postconditions (Success Guarantee)	<ul style="list-style-type: none"> Measurement has been added to the database New battery percentage for the car has been calculated
Main success scenario (Basic flow)	
1. The charging station posts data of a new measurement to the back-end API	
	2.System adds the measurement linked to the charging station socket to the database.
	3. System returns current battery percentage. [End of use case]
Extensions (Alternative flow)	
A: Charging station socket cannot be found	
	2A. [Charging station socket is invalid] The system adds measurement as Smart meter measurement.
	3A. The system returns non valid battery status. [End of use case]

Table 13: Use case - Add measurement

3.10. 3.10. Check charging status

Use case	Check charging status
Primary actor	User
Stakeholders and Interests	None
Brief description	A user can check the charging status of the car being charged
Preconditions	The user is charging their car.
Postconditions (Success Guarantee)	<ul style="list-style-type: none"> The user can see the charging status of the car.
Main success scenario (Basic flow)	
1. The user opens the "my car" page	
	2. The system shows "my car" page
	3. The system retrieves a list of cars and shows it to the user
4. The user selects a car from their car list	
	5. The system shows the data of the car
	[End of use case]

Table 14: Use case - Check charging status

3.11. 3.11. Verify users

Use case	Verify users
Primary actor	Administrator
Stakeholders and Interests	Guest
Brief description	The administrator verifies guests that request an account. He can either decline or accept these requests.
Preconditions	<ul style="list-style-type: none"> A request for an account must be present.

Post conditions (Success Guarantee)	<ul style="list-style-type: none"> • The account of the guest gets verified. • In the database, the data of the guest gets updated. • An email gets sent to the guest.
Main success scenario (Basic flow)	
1. Administrator opens the page "Verify users"	
	<p>2. The system shows new guests who have requested an account on the "verify users" page.</p> <p>The page contains a header component with a hamburger menu. The page shows a container what contains a table with only a header.</p>
	3. The system does a request to the server to get a list of users.
4. The administrator verifies a new account.	
	<p>5. System updates the user as verified and sends an email to the guest who requested the account.</p> <p>[Back to step 2 / End of use case]</p>
Extensions (Alternative flow)	
A: Administrator declines the requested account	
4A. Administrator declines a new account.	
	<p>5A. System deletes the guest from the database and sends an email to the guest that his account is declined.</p> <p>[Back to step 2 / End of use case]</p>

Table 15: Use case - Verify users

3.12. 3.12. Verify cars

Use case	Verify cars
Primary actor	Administrator
Stakeholders and Interests	User

Brief description	The administrator can verify new cars that have been added by an user.
Preconditions	<ul style="list-style-type: none"> • A request for a new car must be present.
Post conditions (Success Guarantee)	<ul style="list-style-type: none"> • The car gets verified. • In the database, the data of the car gets updated.
Main success scenario (Basic flow)	
1. Administrator opens the page 'Verify cars'	
	<p>2. The system shows the page verify cars.</p> <p>The page contains a header component with a hamburger menu.</p> <p>The page shows a container what contains a table with only a header.</p>
	3. The system does a request to the server to get a list of unverified cars.
4. The administrator verifies a new car.	
	<p>5. System updates the car as verified.</p> <p>[Back to step 2 / End of use case]</p>
Extensions (Alternative flow)	
A: An Administrator declines the requested car	
4A. Administrator declines a new car.	
	<p>5A. The system deletes the car from the database.</p> <p>[Back to step 2 / End of use case]</p>

Table 16: Use case - Verify cars

3.13. 3.13. Check charging socket graph data

Use case	Check charging socket graph data
Primary actor	Guest or User
Stakeholders and Interests	None
Brief description	A Guest or User can check how much the car has been charged.
Preconditions	<ul style="list-style-type: none">▪ A car is charging in the charging socket at the charging station
Postconditions (Success Guarantee)	The data has been displayed in the graph on the homepage
Main success scenario (Basic flow)	
1. The user opens the homepage	
	2. The system shows all known charging sockets in the homepage
3. The user clicks on a socket	
	4. The system shows how far the car has been charged and by how much [End of use case]

Table 17: Use case - Check charging socket graph data

4. 4. Other functional requirements

in this chapter, functional requirements that cannot be expressed in the shape of use cases are included.

Code	MoSCoW	Description
FR1	M	The IoT device must be able to connect to the database via some kind of API

Table 18: Other functional requirements

5. 5. Non-functional Requirements

In this chapter all non-functional requirements are described.

5.1. 5.1. Efficiency

The performance with regard to the amount of resources used under the stated conditions.

Code	MoSCoW	Description
NFR1	S	Responses to all user-initiated actions in the web-interface need to be rendered in less than 1 second.
NFR2	M	Graph and socket data should be updated real-time.
NFR3	S	Mails must be send from the system in less then one minute.
NFR4	S	An administrator can send an email from the front-end.
NFR5	M	The system must save the live data from the solar panel
NFR6	M	The system must save the charging data
NFR7	M	The system must save the energy data from the power grid power meter

Table 19: Non-functional requirements - Performance

5.2. 5.2. Security

The extent to which a product or system protects information and data, so that people, other products or systems have the right level of data access that fits their type and level of authorization.

Code	MoSCoW	Description
NFR8	M	Personal user information needs to remain confidential to all parties other than administrators and the user himself.
NFR9	M	Administrator pages must not be accessible for guests or users.
NFR10	M	Personal pages must not be accessible to guests.

Table 20: Non-functional requirements - Security

5.3. 5.3. Reliability

The extent to which a system, product, or component performs specific functions under specified conditions for a specified time.

Code	MoSCoW	Description
NFR11	M	When measurements cannot be added to the database due to server errors/time-outs, it must be possible to retry this on a later moment.

Table 21: Non-functional requirements - Reliability

5.4. 5.4. Usability

The extent to which a product or system can be used by specified users to achieve specified, effective, efficient, and satisfying goals in a specified usage context.

Code	MoSCoW	Description
NFR12	M	The front-end application must be mobile accessible.
NFR13	C	Login for users could be made possible by using NFC tags in mobile phones.
NFR14	S	The user should see his primary car immediately when visiting his cars page.
NFR15	S	The front-end application must be completely scrollable.
NFR16	S	The system needs to show correct error messages about failing to parse data.

Table 22: Non-functional requirements - Usability

5.5. 5.5. Maintainability

The extent to which a product or system can be changed effectively and efficiently by designated managers.

Code	MoSCoW	Description
NFR17	M	Both front- and back-end applications must be able to be tested by executing a single command.
NFR18	M	Both applications must be structured according to the nowadays coding standards.

Table 23: Non-functional requirements - Maintainability

5.6. 5.6. Portability

The extent to which a system, product or component can be effectively and efficiently transferred from one hardware, software or other operational or user environment to another.

Code	MoSCoW	Description
NFR19	M	The applications must be able to be installed on the operating systems described in the action plan.
NFR20	M	The applications must be able to run on the Azure cloud platform.

Table 24: Non-functional requirements - Portability

6. 6. User interface sketches

As a team, we have to work with the code base from the previous team. The UI is not user-friendly because of that the team decides to update the UI.

6.1. 6.1. Base design

Every page is built with a basic template. The basic template contains a header, content container and a menu.

In the left corner of the header is a logo of the website. In the right corner is the icon to toggle the hamburger menu.

The new basic container does not have a dropdown menu as the previous design.

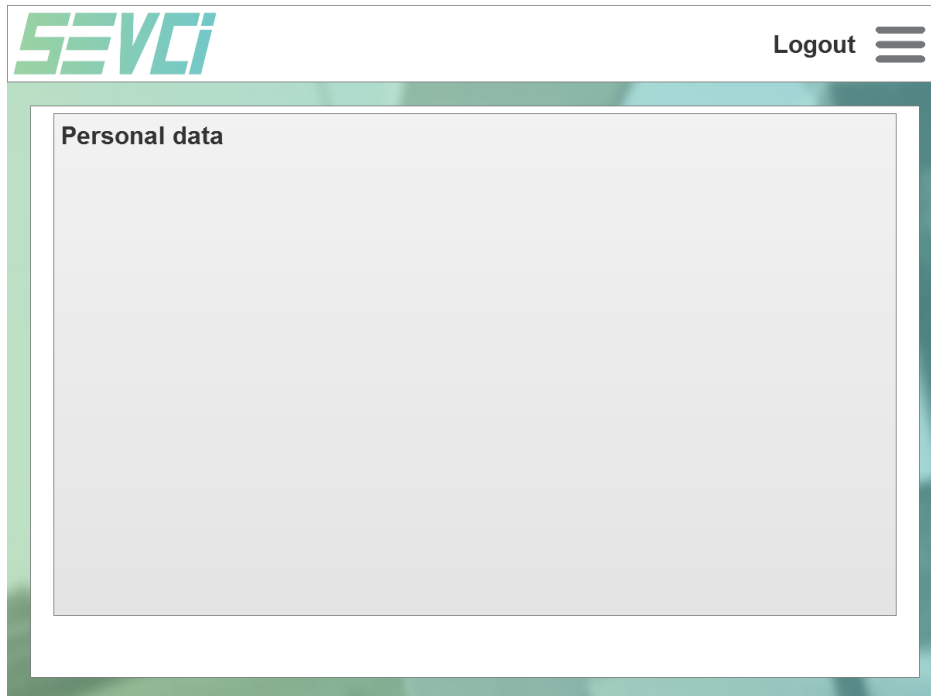


Figure 5: Base container design

6.2. 6.2. Homepage

For the homepage, the team decided to add the sockets at the top of the container. At the bottom of the container is a graph with the current energy flow.

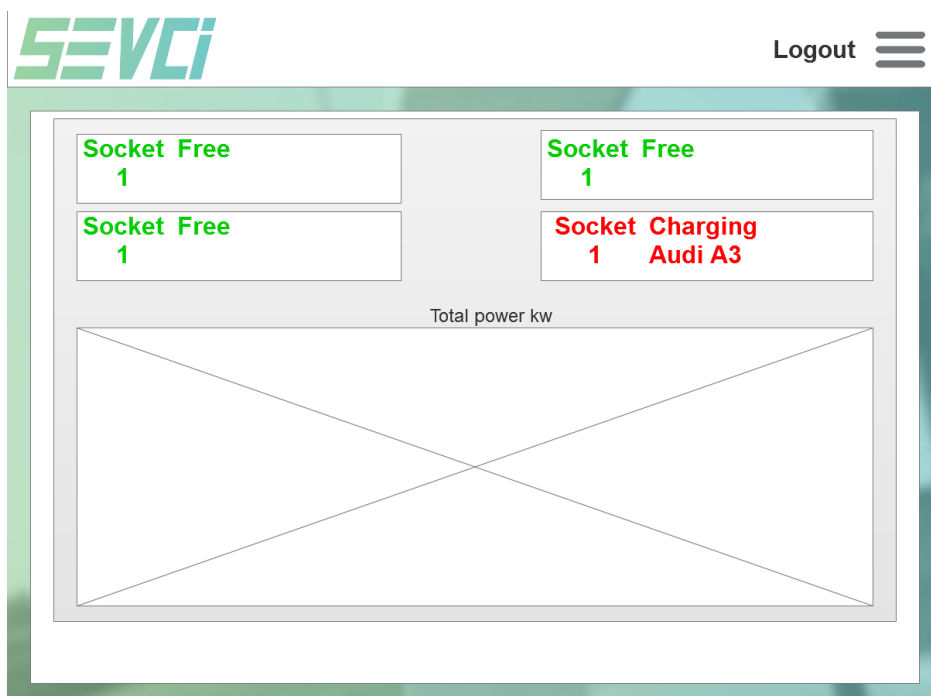


Figure 6: Homepage

6.3. 6.3. About

The page 'About' contains information about the website and the project. In the old design, the names of the developers put at the right side in the container.

In the new design, the team moved it to the middle of the page.

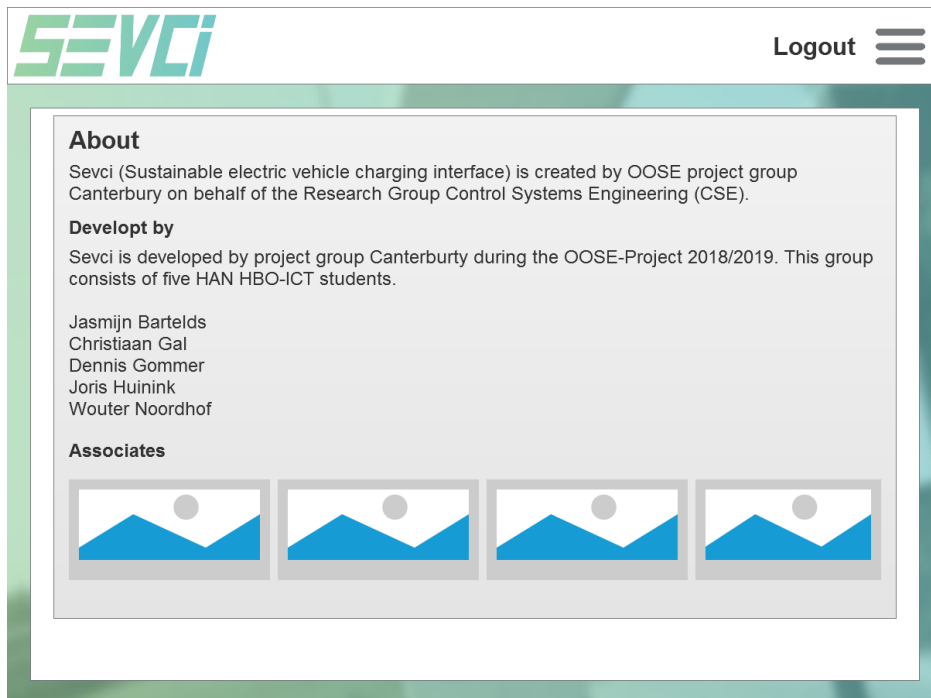


Figure 7: About page

6.4. 6.4. Login

The page 'Login' is a functional page for the user where he signs in.

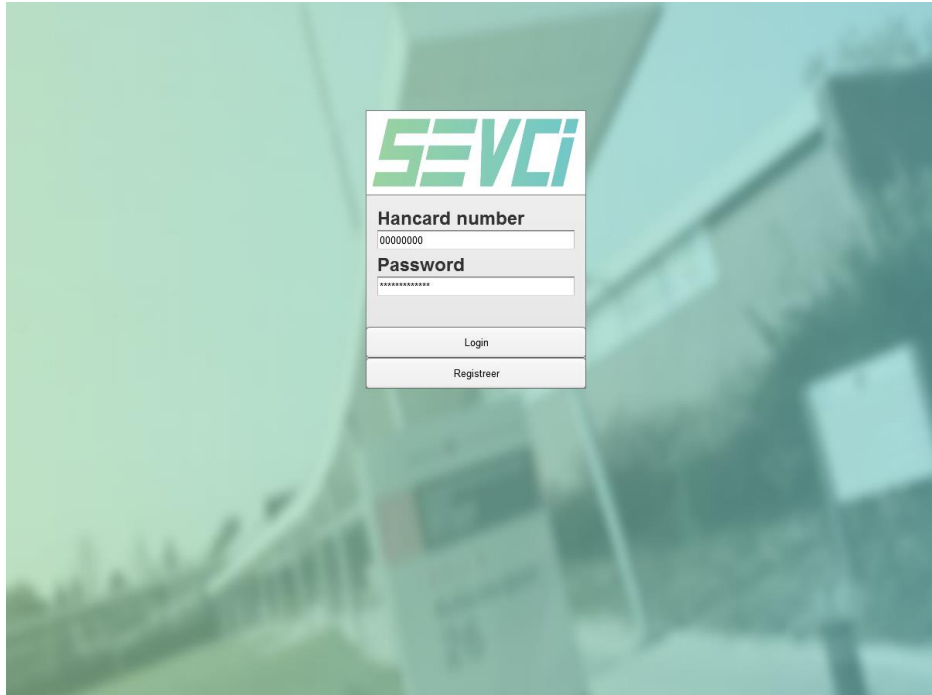


Figure 8: Login page

6.5. 6.5. Register

The 'Register' page is used for the user registration. In the old design, the user read from the left to the right in different columns. In the new design, it is a list layout.

Figure 9: Register page

6.6. 6.6. My cars

The page 'My cars' is used to show the user cars. For this page is a table layout used. In the new design, it is a list layout.

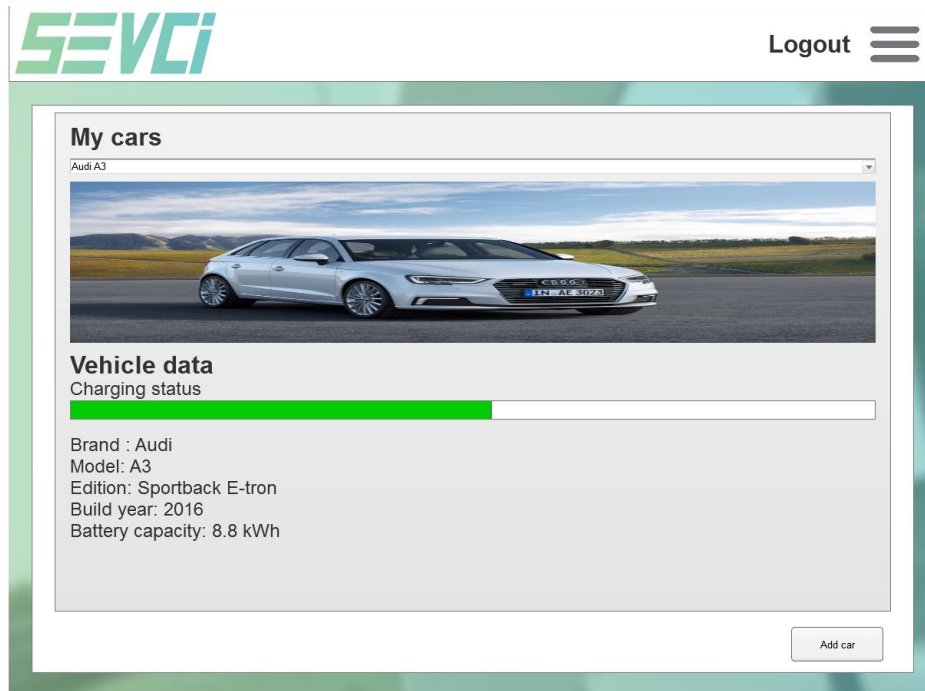




Figure 10: My car page

6.7. 6.7. Add a car to user

The page "Add a car to user" is used for adding a car to the existing list of cars in the database. Add a non-listed car is when you cannot find the car on the list. Confirm is when you want to add a car to our list.



Logout 

Add a car

Find your car

Brand	Type	Edition	Year	Capacity in kWh
Audi	A3		2013	85
Audi	A3	Sportback E-Tron	2016	8


Add non listed car


Confirm

Figure 11: Add a new car page

6.8. 6.8. Add a new car

The page 'Add a new car' is used, so that the user can add a new car to the system. In the old design, the text was centered in the middle. In the new design is it centered to the left.



Logout 

Add a new car

Brand

Audi

Model

A4

Edition

Sport cars

Build year

2019

Battery capacity

50

Save

Figure 12: Add a new car page

6.9. 6.9. Battery fill in

The page 'Battery fill' in is used so the user can add their battery voltage to the system. In the old design was the car details a table layout used. In the new design, it is a list layout.

SEVCI Logout

My cars

Audi A3

Battery status

To give you an estimated battery charge percentage fill in the current battery percentage below.

Current battery %

50%

Vehicle data

Brand : Audi
Model: A3
Edition: Sportback E-tron
Build year: 2016
Battery capacity: 8.8 kWh

Save

Figure 13: Battery fill in

6.10. 6.10. Verify users

The page 'Verify users' is used to the admin so he can accept or reject people from the system when they registered. In the old system, the first two columns contained the accept and deny buttons. In the new UI, it has been decided that the accept and deny buttons will be in the last two columns.

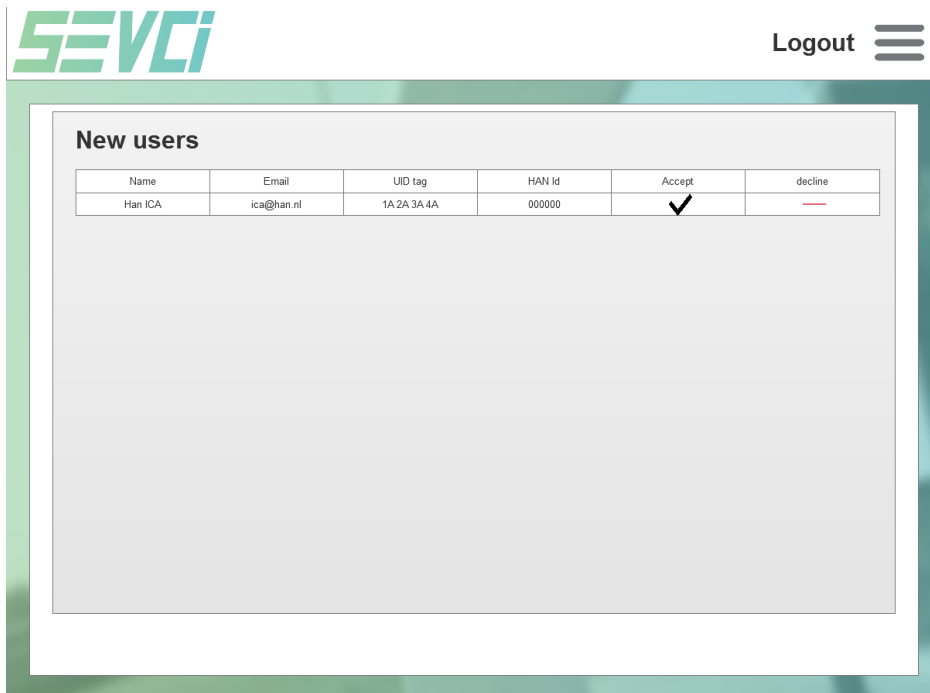




Figure 14: Verify users page

6.11. 6.11. Verify cars

The 'Verify cars' page is used because the admin can approve new cars. In the new UI, the accept and reject buttons moved to the right



Logout 


Verify Cars


Brand	Car model	Edition	Build year	Battery capacity	Accept	decline
Audi	A3	2	2018	8KW	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Figure 15: verify cars page

6.12. 6.12. Account info

The page 'Account info' is used for the user personal details. In the old design, the user read from the left to the right in different columns. In the new design, it is a list layout.



Logout 

Personal data

Fullname

HAN De ICA

Email

ICA@HAN.NL

New password

Repeat new password

Static data

Han ID : 0000000

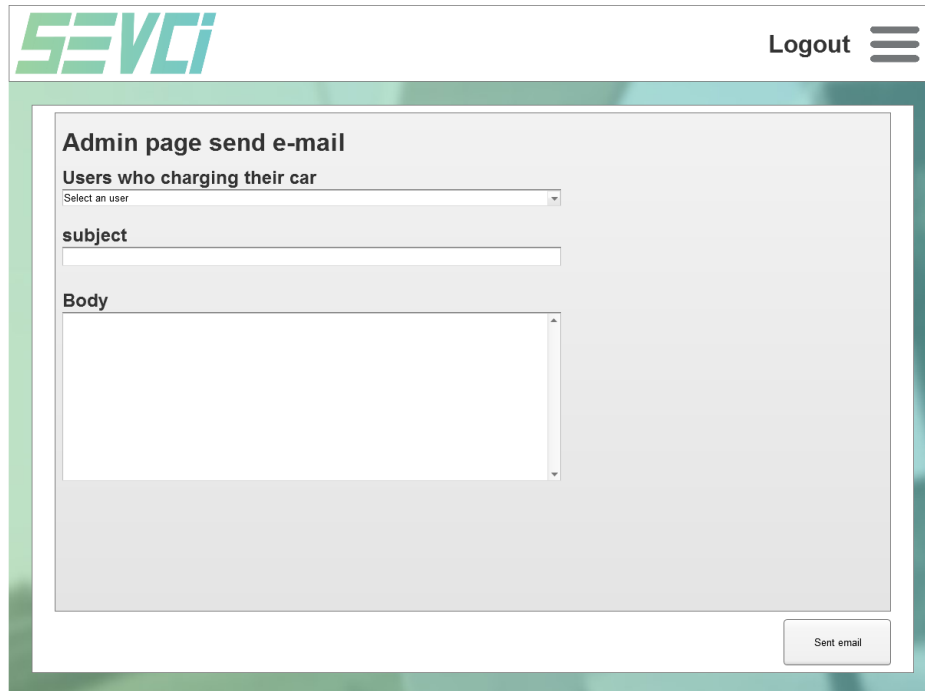
UID Tag: A1 A2 A3 A4

Save

Figure 16: Account info page

6.13. 6.13. Admin page send email

The page 'send email' will be used to send an email to a user that is charging his car. In this design, the team created a simple design to send an email in a matter of time. The dropdown list contains users who are charging their car. The subject field is used for the email subject and the body text area will be used for the email body.



The screenshot shows a web application interface for sending emails. At the top left is the 'SEVCI' logo. At the top right is a 'Logout' button next to a hamburger menu icon. The main content area is titled 'Admin page send e-mail'. It contains a dropdown menu labeled 'Users who charging their car' with the placeholder text 'Select an user'. Below this is a 'subject' text input field. Underneath the subject field is a 'Body' text area. At the bottom right of the form is a 'Sent email' button.

Figure 17: Admin page send email page

7. 7. Resources

- Oracle. (2016, 6 januari). Windows System Requirements for JDK and JRE. Geraadpleegd op 9 mei 2019, van https://docs.oracle.com/javase/8/docs/technotes/guides/install/windows_system_requirements.html