# Technical Documentation

**Team - Members**
**Bart Salfischberger (500839321)**
**Rick van Siepelinga (500854491)**
**Miguel da Silva Crespim (500853533)**
**Stef Kan (500856121)**
**Silas de Jong (500806288)**

**Date: January 9, 2022**
**Class: IS201**
**Team: CCU-1**
**Version: 1.0**

# Table of Contents

# Introduction

The goal of this project was to make an application to manage greenhouses. We have provided the users with tools, which they can use to reduce the total co2-level in the greenhouses. Our application also provides different levels of customization to the greenhouse, ranging from building teams, creating or inviting users, creating notes, and sharing content with other users. The project client provided our team with the API, to access the greenhouse sensors, it was our responsibility to retain the data in the back-end.

# Product Vision

Our vision for the product was simple, to create an easy-to-use web application that is very minimalistic in design. The challenge was to keep the site clean while still being able to show lots of data. This is done by the use of graphs in our application. We also wanted to give the user a few personalization options, a user can change their profile picture and order of graphs. Another important aspect was to use a form of multitenancy in the application.

With this multitenancy a user can be a member of multiple teams (with different roles) with only one account. The multitenancy also adds to collaboration instead of competition, and makes it easy for an expert or academic personnel to collaborate. The end goal of this application was to lower co2 in each greenhouse. We wanted to boost the feel of collaboration instead of competition.

# Epic stories

As for our most important epic stories from this project, we have chosen these three:

- **The Graphs**
- **The Authorisation system**
- **The Note page**

The note page provides the standard create, read, update and delete operations, which allow users to do all the required tasks for note keeping. One must have for our note page was the share function, which gives users the possibility to share their notes with members of other teams. If the other team is interested in reading these notes, they can find it under the note category filter in question. Which brings us to the final feature of this page, the note search filter function. By entering the title of the desired note, or the content of the note, we can filter through the list of notes meeting that criteria.
**(Our focus for this software quality attribute was functionality and efficiency)**

**For the note page we had created these criteria:**

- As a user I can successfully create, and edit a note. To document my research process.
- As a user, on page start-up, I want to see all the notes which belong to the team and shared within my scientific role. **(Important)**
- As a user I want to be able to search through the notes list using a search bar, to find my desired note.
- As a user I want to be able to share my notes with other teams, so others can see my research. **(Important)**
- As a user I can filter through note category's (Scientific roles:  botany, hydrology, etc), to find a particular note **(Important)**

The authorisation system acts as the security within our project. By checking our users with each step, front and back-end, we make sure that the users are properly authorized to perform important tasks. This is made possible by the use of the JSON Web Token.
Each request gets filtered by our request filter, which checks whether the user has the right authority. We made sure that the JWT didn't contain any sensitive data like the user's password or other security breaching details. Another security measure we implemented was setting the expiration time of the generated token to 3 minutes, which ensures that after each interval we re-authenticate the user, by refreshing the token.
**(Our focus for this software quality attribute was maintainability and reliability)**

**For the authorisation system we had created these criteria:**

- As a user, I don't want to login every time a token expires. **(Important)**
- As a user of a team I should not be able to request any resources I'm not authorized for. **(Important)**

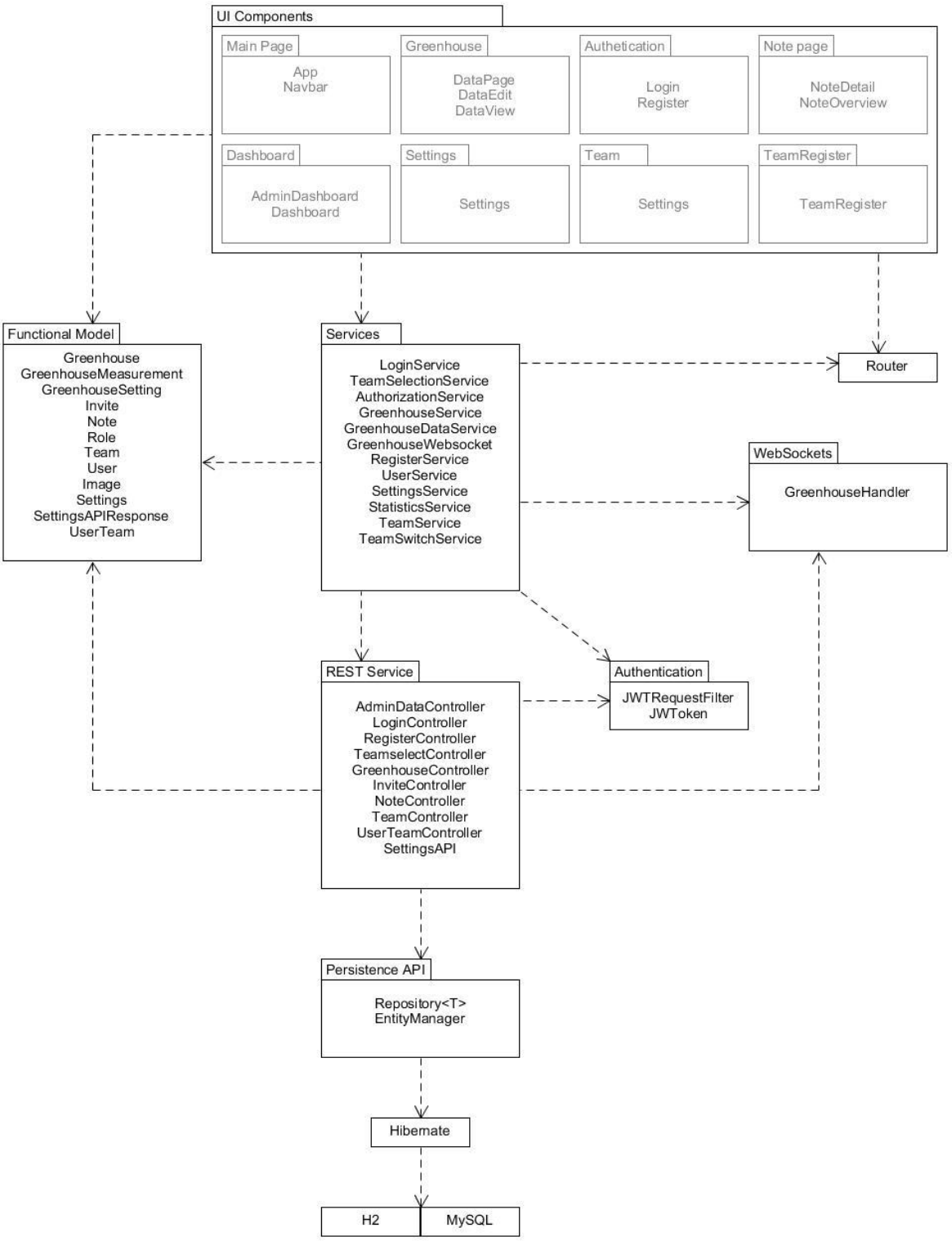
As for the Graphs epic story, we have used the d3.js library for implementing the graphs with the provided greenhouse API. We did this because the 3d library allows us to build exactly the type of Graph that we needed to represent our greenhouse's data. Not only did we make an easy to understand graph, but we gave users (with the right authority level) the tools to make adjustments to the greenhouse. As a safety net, we have restricted the user from changing the greenhouse settings to dangerous levels, which they will be alerted by a notification. We also save our greenhouse's historical data, which we use to calculate the hourly and per minute average.

**(Our focus for this software quality attribute was functionality and  usability)**

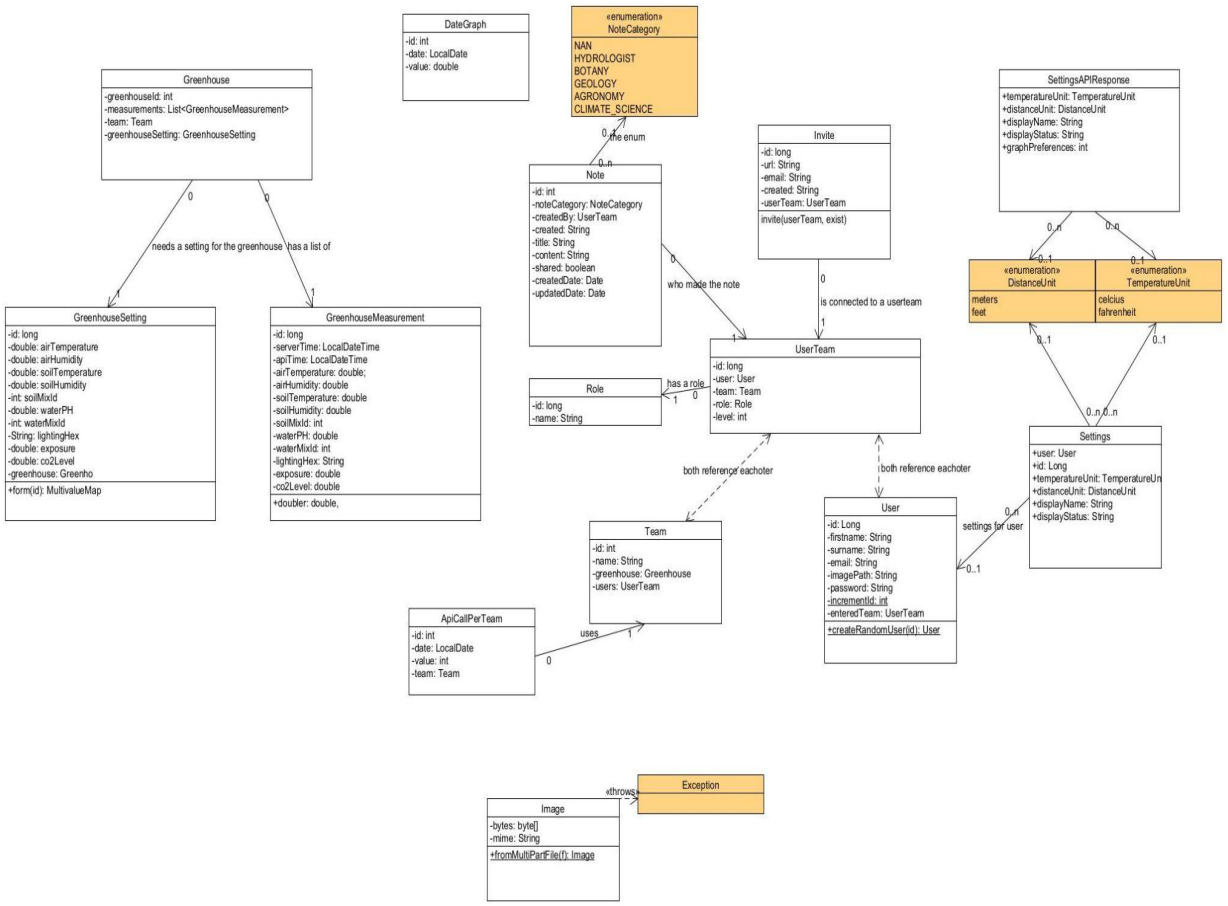
**For the Graph epic stories we had created these criteria:**

- As a user I want to have a clear overview (graph) of given sensor data, using the following graphs: Histogram, Line graph, Area, Pie chart.  **(Important)**
- As a user I want support for multiple charts in the given component.
- As a user I am able to send data to the devices in the greenhouse. **(Important)**
- As a user, I want to view the hourly changes in any given data from the greenhouse.

# Navigable class diagram

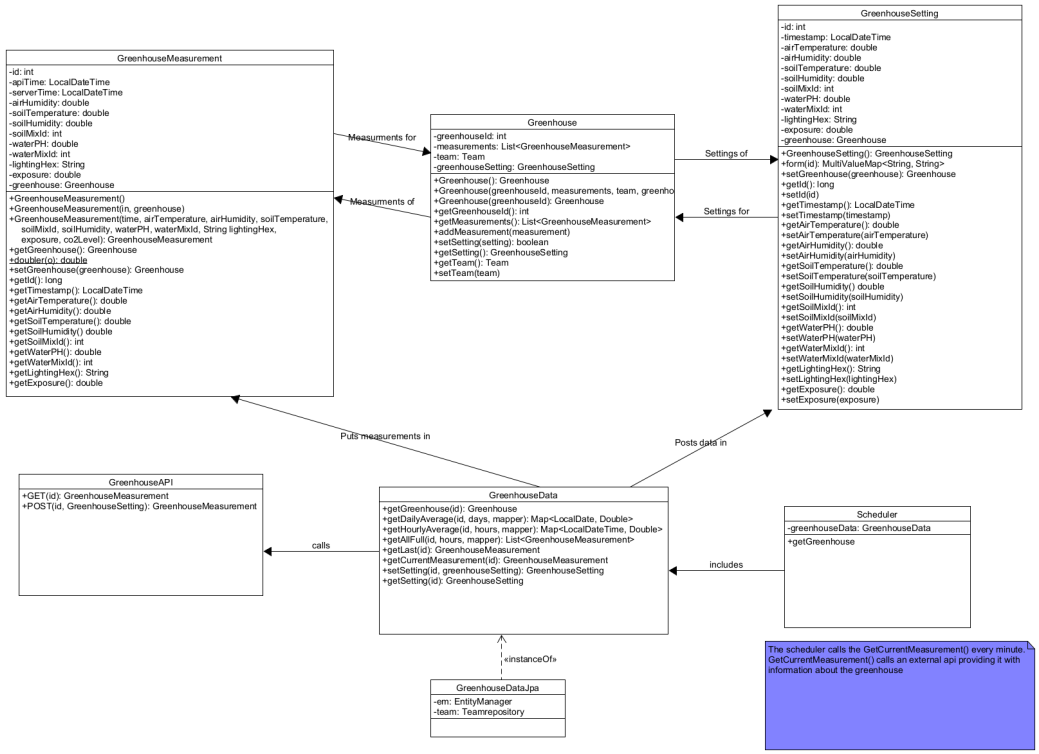



This diagram shows the full stack layered logical architecture of our application.
The front end is set in ui components. Every class is set in their right functional model.




The text on the arrows share their purpose on the other entity.
The arrow tips of the associations indicate ‘navigable relations’. Some are bi-directional, others are uni-directional.

- For this model, we didn’t include the constructors and the getters and setters.
- This diagram only includes the main entities, attributes and their operations.
- Every attribute has a getter and a setter that it can use.
- Bi-directional navigability: The two entities have relationships with each other, and both use each other.
- Uni-directional navigability: One entity has a link with each other.




The scheduler calls the getCurrentMeasurement() function every minute. This function makes a call to an external API and puts the data in a GreenhouseMeasurement assigned to the correct greenhouse.

The API is also called when setSetting() is called. This makes a post request and assigns the new settings to the appropriate greenhouse. The correct greenhouse is specified in an id given along as a parameter in both the function and the request.

# Design choice

During the development of a product, important choices have to be made that affect the development process and the end-product. Many of the choices made during the design and development phase of our dashboard have made such drastic differences to us and the product. This paragraph is dedicated to pointing out some of these decisions and the reasoning behind them, as well as reflecting on these choices to see how we can improve the process for future projects.

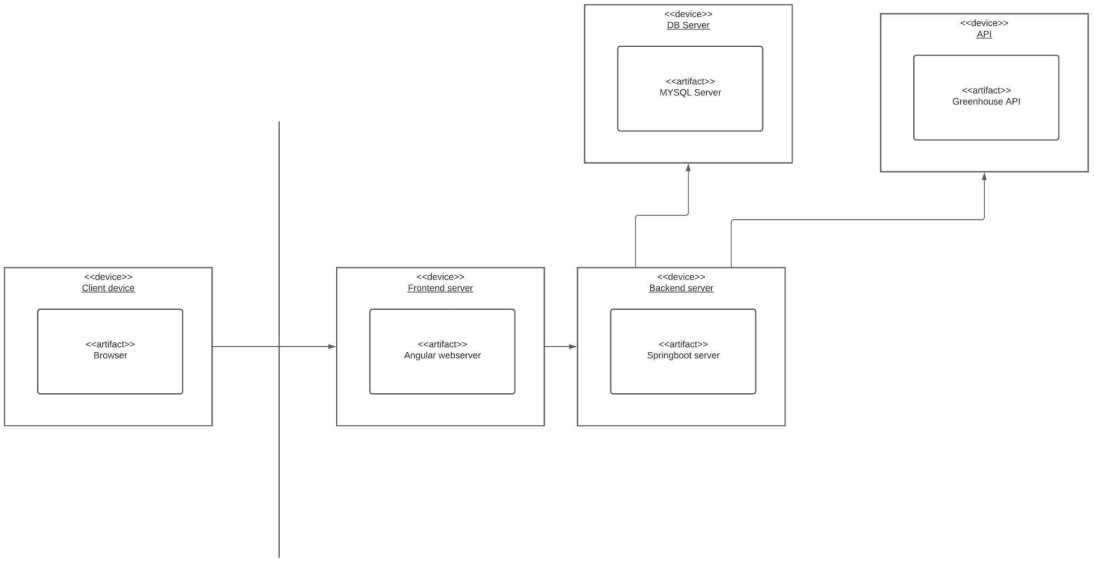
One of the most important documents for effective and efficient product design is the Moscow document. In this document you and your team decide which features will make it into the end product, which feature will be excluded from it, and which features could optionally be implemented when there's a surplus of resources.

Since our application is a web-dashboard application we decided to design a minimal and clean interface that doesn't distract from the main product, the dashboard. Our interface has one sidebar, that only takes in a small portion of the interface, and a main application area. When on the main dashboard page, about 67 percent of the width of your screen is used for the dashboard. We decided to change our philosophy to quality over quantity. This means that we provide our users with a working application with fleshed out essential features, and some nice extras like allowing the user to customize their profile.

As a researcher it's essential that you're able to make notes and share them with other researchers. For this reason we implemented a note-keeping system that allows research to document their findings. But we think it's also very important that researchers have the ability to share their notes with other researchers within the same field. This way knowledge can be shared and enhanced by other gifted minds. The note-keeping system allows users to create, delete, edit, share, look for, and filter notes.

We decided against building a translation function into our application. Even though it is a nice feature to have for international scientists, we decided that the overload of domain-specific language would be very difficult to translate if not done by experts. And since most scientists are involved in international communities, we think it's fair to assume they also have a firm grasp of the English language.

# Deployment diagram




We used an Angular server to host the client side website. We chose Angular because this framework would give us the easiest way to create a dynamic website.

For the backend we used spring boot because this would make working with java easier. This would also make working with http-requests easier as we could use spring boot dependencies like spring-boot web.

The frontend and backend servers are split because the two processes are different. The frontend runs node and the backend runs java. Splitting these processes would make the most of the hardware that it runs on.

The backend server is also connected to a MySQL database server. The database stores the user-data, team-data and past greenhouse data per team. The user-data and team-data are stored in a database because this data needs to be persisted. The greenhouse data is saved because there are graphs that show past data and the api doesn't store past data.

We used MySQL because our application relations were important. A NoSQL server would not allow relations. The MySQL process has its own server because this way you could make the most of the hardware that it runs on.

There is also a connection with an API. This is accessed through the backend server and used to get the current greenhouse-information and set new greenhouse-information.

The API has its own server because it is mostly handled by external parties. In development it uses a black box and in production it uses a greenhouse. Having the API be an external server makes it easy to switch between a black box and a real greenhouse.

# Analytical Reflection

While we as a team are happy with what we achieved during this project, there are still enough improvements we could've added to really make this application a ten out of ten. This document should already tell you enough about how working on the application itself went, but in this paragraph we really want to reflect on our choices, what did we do which we're proud of, and what could we have done better?

## Our design process

To start off, I want to explain how we started designing this project and how we continued to make decisions for the designs through the whole project.

On the day we started the project we all came together as a team and started brainstorming about the ideas we as individuals had for the project, and how we could combine those ideas into making the best possible application we could. We basically held our own kick off meeting. We looked through all the requirements the clients had, and decided to incorporate those into our frontend design. Once we knew what the requirements were and had a good idea of how we wanted our website to look like, it was time to start thinking about the back-end side of the project.

While thinking about the backend we had to consider a lot of different factors. How were we going to ensure that the application was going to be fast, that the application will run on every device, that the application won't crash. There was so much we had to take into consideration it started to feel like a lot, yet we took our time and discussed all our concerns with each other.

Once we were done with making our plans for the application, it was time for us to start working on the actual application.

## The good

First things first, we're really proud of how our graphs look and how our dashboard functions. As a user you can decide what graphs you want to see on your dashboard, and you can customize them as well. The way we implemented this, it's also future proof! Even if you add more graphs, the user can decide which one they get to see on the dashboard.

We're also happy with our team selection system. The way we designed our application is that you can be part of multiple teams, and that you can switch from a team at any given moment. This was made possible by editing our JWT (JSON Web Token) whenever someone switches from team, and then the application gathers the data from the different team. For the future, we might want to check into making sure we're giving even less data with the JWT.

In conclusion, we're definitely happy with some of the features we've designed, yet there is always room for improvement.

## What could have gone better

There is definitely still room for improvement in our application, and we'll elaborate on down below.

Our first point, further future proofing. Our application right now has only the graphs, and the features that the clients asked for, yet we don't know if they want more features in the future. Our application isn't fully ready for new features yet, so that would be a bit of a problem. We don't think it will be a problem with how the black boxes work, but it could be something we can think about.

To make our application more future proof, we could've made it so that the admin of the application can also implement new graphs.

Or something we didn't implement but did think about was the progress page. The reason we didn't implement this feature was because we received the request for it a bit too late. If we had more time for the project we would've probably implemented it in a similar way we have implemented the note page right now.

For this project we use the Angular framework, this means we get to work with TypeScript. The best thing about TypeScript is getting less errors while developing due to the use of types. The problem is we casted too many objects as "any" instead of using types.

## Conclusion

We went into the design process with a lot of experience because of the past two projects we had at this study, and because of that we had a clear focus for the application.
While we are happy with the result of the application, there are definitely some features we would've liked to implement, or improve on.

We're happy with the choices we made for this project, but can acknowledge that there are definitely improvements we could've made to the design.