# *TECHNICAL DESIGN DOCUMENT*

*Project EWA – Sustainability Dashboard*

*Callum Svadkovski, Saif Rashed*

# Table of contents

# TECHNICAL DESIGN DOCUMENT

*Overview*
*The Technical Design Document describes the system requirements, operating environment, system architecture, database design, input formats, output layouts, software design, and an analytical reflection.*

## 1 INTRODUCTION

### 1.1 Purpose and Scope

This document serves as a guide for the software development process. From gathering the initial software requirements, creating use cases, the software system design and architecture (ERD and class diagrams). the documentation serves as an instruction manual for the software which has been developed through the course of project EWA.

### 1.2 Project Summary

The Green Office HvA (GO HVA) is a department within the HVA that is committed to sustainability within the HVA and brings staff and students together for this purpose. GO AUAS initiates and monitors sustainability projects. GO HVA consists of employees and students.

#### 1.2.1 System Overview

The Sustainability Dashboard provides structural insight into the extent to which sustainability is represented in all parts of the AUAS and thus provides guidance and structure for further integral sustainable development of our educational institution. The Dashboard is therefore, a method in which the status of the AUAS with regard to sustainability can be measured and inspires new activities. This allows plans to be monitored and new plans initiated to achieve the AUAS sustainability goals. The dashboard contains 4 pillars: education, research, business operations and organization There are +22 goals and +51 detailed sub-questions; Currently there is only an Excel sheet in which the questions and answers must be included.

#### 1.2.2 Design Constraints

The Sustainability Dashboard shall be managed by the GO HvA team. The system will be a tool used internally within the organization of AUAS. The dashboard must therefore form its metrics based on the contents of the delivered excel sheet. As described in section 1.2.1 the pillars consist of multiple goals and so on. Therefore, the system is restricted by design to a limited hierarchical system. The scoring will be based on a numeric value which in turn will allow us to calculate the averages. The averages will start at the smallest possible group (programs) and build up to compose a global average of the organization. This means that each user beside the admins should and must be categorized by their faculty and program to facilitate this scoring method.

### 1.2.3 Future Contingencies

The problems that could arise regarding the design of this system might stem from multiple places. First of all, we have observed an inflexibility regarding the pillars. These are constants and therefore not dynamically updated. This means that a change in naming can cause problems in the design. Secondly, the surveys do not have a visibility option. This means that a faculty user could fill a survey made for organizational users. This is one of the larger contingencies because of the potential damage in the metrics because of some accidental filling of surveys. A workaround would make use of a visibility option for surveys but this would require a more extensive selection of roles. The combination of the two would allow for a more robust surveying design.

## 1.3 Points of Contact

The key points of contact (and alternates if appropriate) for the GO HvA system development effort.

- Gideon Bosses (g.h.bazen@hva.nl) Change Agent Sustainability FDMCI
- Pavel van Deutekom (p.van.deutekom@hva.nl) GO AUAS
- Karthik Srinivasan (k.srinivasan@hva.nl) POD

## 1.4 Project References

A bibliography of key project references and deliverables that have been produced before this point.

- **GitLab Repository**
  - https://gitlab.fdmci.hva.nl/se-ewa/2021-2022/stb-1
- **GitLab Boards**
  - https://gitlab.fdmci.hva.nl/se-ewa/2021-2022/stb-1/-/boards/4023
- **Installation Manual**
  - https://gitlab.fdmci.hva.nl/se-ewa/2021-2022/stb-1/-/blob/main/README.md
- **Project Manual**
  - https://dlo.mijnhva.nl/d2l/le/content/354055/viewContent/896082/View
- **Sustainability Webtool briefing**
  - https://dlo.mijnhva.nl/d2l/le/content/354055/viewContent/904600/View

## 1.5 Glossary

A glossary of all terms and abbreviations used in this document.

| | |
|---|---|
| AUAS | Amsterdam University of Applied Sciences |
| FDMCI | Faculty Digital Media and Creative Industry |
| GO HvA | Green Office HvA |
| ERD | Entity relationship diagram |
| Faculty | A categorical divide within the AUAS educational offering |
| Program | Refers to a study within a faculty |
| Service | Often referring to the Angular Service classes |

# 2 PRODUCT VISION

In this section we summarize the product and its most important realized features. We describe the target users and their interests. We determine the position of the product in the market relative to alternatives and competitors.

## 2.1 Features

To start of we shall describe a few of the most important epics that have been realized within the application. We shall further explain the acceptance criteria for each one of the epics. A description of the software quality attributes will be attached.

**1. GO AUAS must be able to fill in and adjust questions**
This epic story forms the basis of the application. The acceptance criteria are as follows:

1. Admin can create a survey with a form
2. Admin can update a survey and change faculty
3. Admin can assign descriptions to scoring values from 0 to 5
4. Admin can assign pillar to a survey

*Usability* is the key Software Quality Attribute regarding this epic. The execution of this story was largely dependent on the correct execution of the following:

**Learnability**: the steps to creating a survey should be easy to use and remembered.
**Operability**: the system should be in a safe and reliable functioning condition.

**2. GO AUAS must be able to generate overviews**
This epic story gives function to the application for GO AUAS and is therefore an important piece for a successful end result. The acceptance criteria are as follows:

1. Admin has insight on the global, faculty and program scoring averages.
2. Admin has access to visualized data with means of graphs.
3. Admin has access to data organized across time.

*Maintainability* is the key Software Quality Attribute regarding this epic. The execution of this story was largely dependent on the correct execution of the following:

**Analyzability**: User input can be analyzed and processed by the GO AUAS team.
**Reusability**: the scoring system is generic in its nature and can be used for different surveys.

## 2.2 Target users

To pin point the target users we can look to the different goals GO AUAS tries to achieve. The foundation of all these goals built upon the described pillars; education, research, business operations and organization. Each of these pillars have a different group associated

with it.

**Education**
Here we can expect the target user to be *program coordinators* within the different faculties. They are responsible for the educational goals within AUAS.

**Research**
For this pillar we can expect *researching parties* within the AUAS to handle the questions. These parties should have access to employee, partnerships and strategy data.

**Business operations**
The business operations can only be answered by those working in the *logistics* section of the AUAS.

**Organization**
The organizational surveys are only for those that are responsible for higher level decisions. We can expect *management* to fill in these.

## 2.3 Market position

The sustainability dashboard has a key function of receiving input data through surveys and displaying organized and visualized data as an output. We can therefore regard the application as a survey administration software. This means we can set it aside two similar players in the market.

**Google Forms**
The Google Forms service has undergone several updates over the years. Features include, but are not limited to, menu search, shuffle of questions for randomized order and much more.

**Microsoft Forms**
Microsoft Forms is an online survey creator, part of Office 365. Released by Microsoft in June 2016, Forms allows users to create surveys and quizzes.

The dashboard gives AUAS much more control over the data and allows for more creative data visualizations. Furthermore, it should be clarified that the above examples do not allow integration with the AUAS SIS platform. Using these solutions can become very limiting in a later stage of development.

# 3 DATABASE DESIGN

This section should describe the design of the database (NoSQL). Additional information may be added as required for this particular project.

## 3.1 NoSQL design

We chose to use MongoDB Atlas platform for the database platform. The NoSQL approach we chose to use was done because of multiple reasons:

- The delivered datasets corresponded optimally to NoSQL JSON structures.
- Fast querying and aggregation.
- Angular integrates with MongoDB JSON because of JavaScript.
- Flexible design (may be updated easily across development).
- Powerful aggregation queries which play a role in the statistics.
- Can facilitate large amount of survey data

To visualize and model our database design we chose use both a typical relational diagram and a UML class diagram. The relationship is far simpler because we use NoSQL but have been allowed to do so by the POD.

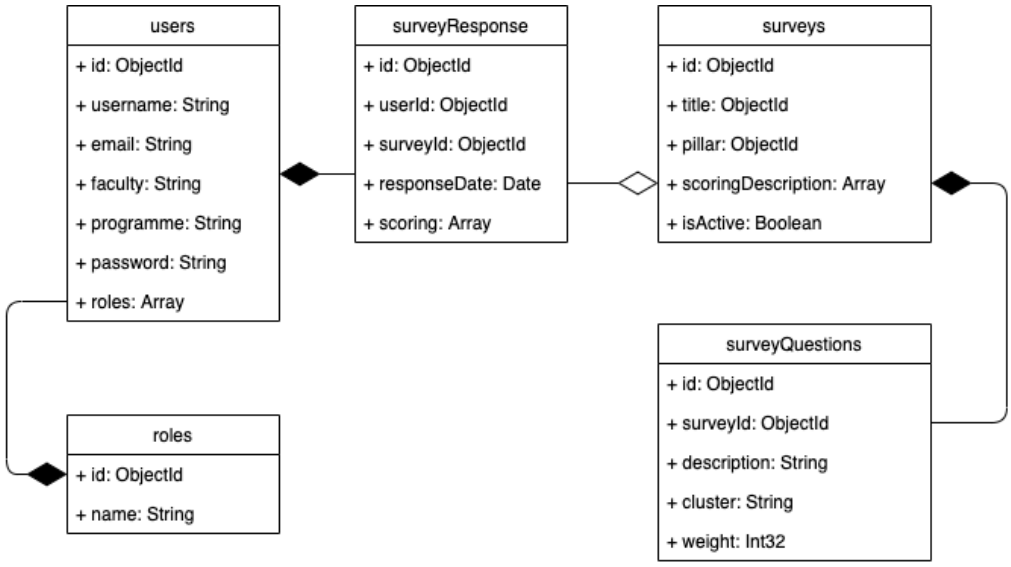



**Figure 1: NoSQL data model using UML**

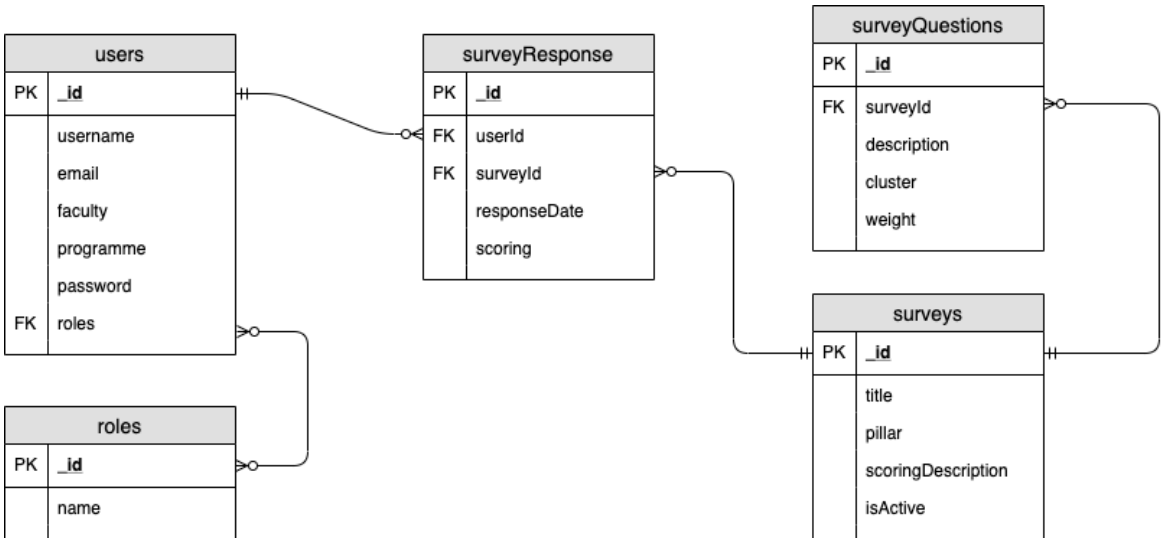



**Figure 2: Data model using a typical ERD diagram**

# 4 SOFTWARE DESIGN

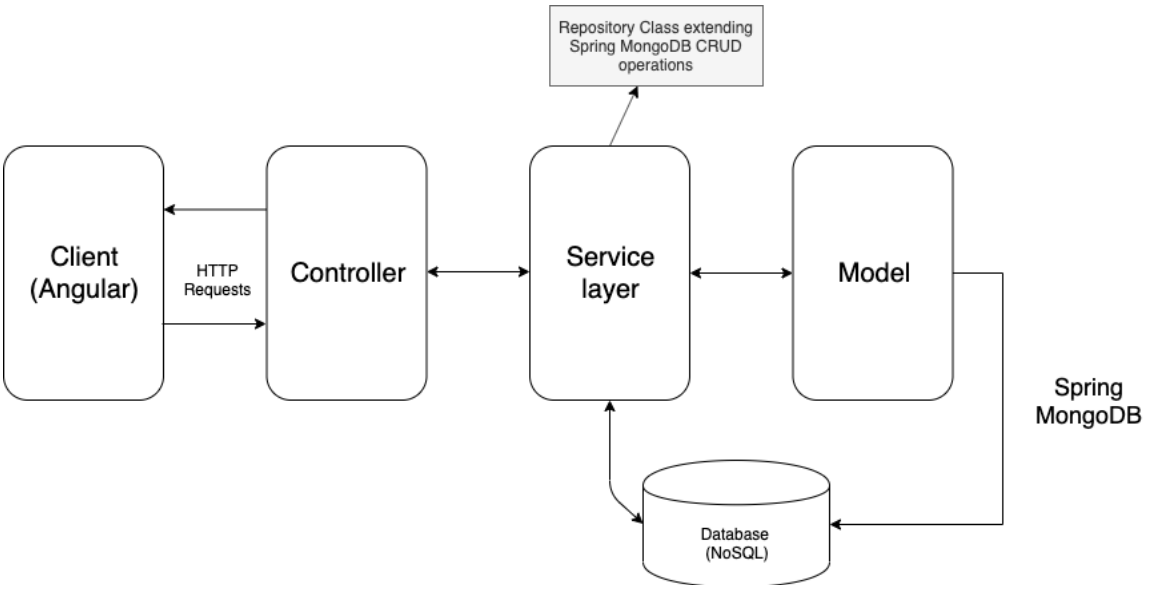
This section provides the information needed for a system development team to actually build and integrate the hardwarecomponents, code and integrate the software modules, and interconnect the hardware and software segments into a functional product.

## 4.1 Software Architecture

This section should provide enough detailed information about logic and data necessary to completely write source code for all modules in the system. An introducing diagram will consist of the global flow within the app. This will give an impression of data travelling across the app. Following up is a detailed UML-diagram for both the client and server.

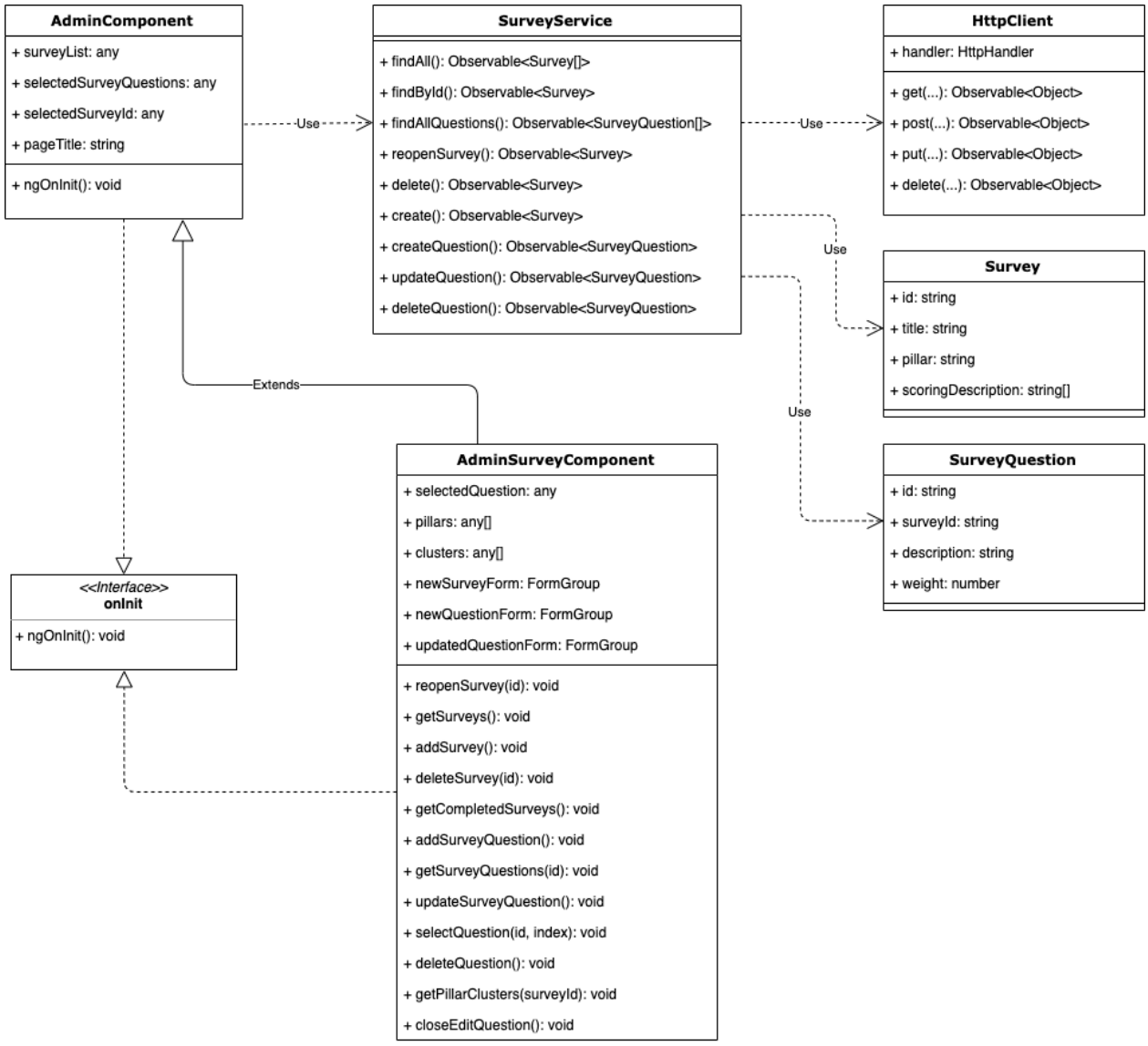
### 4.1.1 Flow

This flow diagram shows the global workings of the application. It shows a decoupling between the client and server. The client makes use of HTTP-requests to connect with the controller. The controller will in turn interface with the model using the repository classes which act as an intermediate. The model contains the Spring MongoDB API and is used to manipulate the database.



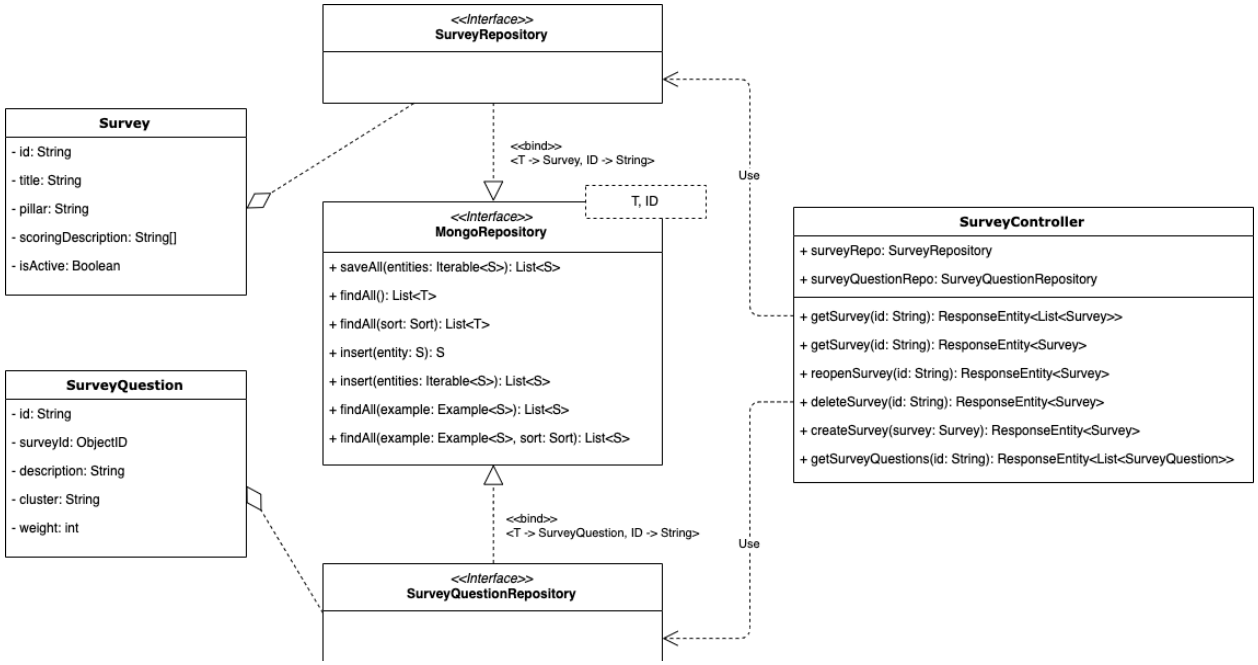
### 4.1.2 Client

The client-side has a goal of receiving input data and displaying output data using designed interfaces. We will approach this explanation in a top-down manner. The initial class here is the HTTP Client. This is an inbuilt package within Angular that helps us to create requests. The survey service makes use of these methods to send out different kinds of request to a controller. The service is used in the parent admin component. We use a parent admin component because the different child admin pages have similar functionalities. The admin survey component extends from this parent component and is responsible for displaying the view and handling the user input.



### 4.1.3 Server

The server-side has the goal of interacting with our model. The flow of this system starts at the HTTP request that is picked up at the controller. The spring boot mappings will associate a route with a defined method. It is here that the system will start ‘talking’ with the repositories. The repositories are separated based on its associated entity. All our repositories extend from a parent class that contains all the generic CRUD methods that is needed for handling MongoDB queries. That explains why we have a few empty repositories in our current system. This does not mean that the empty interface classes are useless but that this architecture allows us to make custom queries for each entity which we currently don’t utilize.



# 5 ALTERNATIVE DESIGN

This section elaborates on the challenges and alternative solution options of an alternative design theme, which proved relevant in this case.

## 5.1 Challenges

During development we have as a team encountered multiple challenges that have resulted in a loss of productivity. We think these problems are possible to fix as we shall describe them in section 5.2. First of all, we shall state the most influential of these obstacles.

**Spring Boot MongoDB API allows for aggregations but these tend to become long methods and hard to read.**

**Angular http request service methods return observables and therefore don't hold a global state within the application.**

**Routing and app modules have slowed productivity because of its constant need to be updated.**

**The need to have separate deployment environments for a single purpose application seems redundant.**

## 5.2 Alternatives

The above obstacles have their root cause in the choice of technology. If we look at alternatives, we can find better solutions that scale more easily. First of all, it should be noted that Angular has been a reliable technology but we have seen technologies like Next.js come up with promising solutions. Next.js is built on React but handles the complexities of routing and in combination with Redux allows for a global state. The use of Next.js allows us to combine the client and server in one express server. This gets rid of the requirement for multiple deployment environments.

Unfortunately, the change in core technologies would be a time expensive task but there are improvements to be found in the current design. One of these improvements would be the design of our service. We can achieve global state by changing the design of our services to encompass a property that caches data. We currently return observables but this seems to remove our ability to hold global state. Persisting data across all components is possible if we are able to hold data within a property that lies above all components. Considering the service is placed at the root element of the application it makes sense to use this method as a solution.

```
logout() {
    // remove user from local storage to log user out
    localStorage.removeItem( key: 'currentUser');
    this.currentUserSubject.next(null as any);
}

findAll() {
    return this.http.get( url: `${environment.apiUrl}/api/auth/users`)
}

findById(id: any) {
    return this.http.get<User>( url: `${environment.apiUrl}/api/auth/users/` + id);
}

deleteById(id: any) {
    return this.http.delete( url: `${environment.apiUrl}/api/auth/users/` + id);
}
```

**Figure 3: Current service method design**

# 6 DEPLOYMENT

It should be stated that the choices that have been used for the deployment are open for improvements. The current deployment strategy has been based on least cost. To clarify the deployment, we will list the used platforms and to visualize this we will add a deployment diagram:

**Client container**
*Express Server consisting of an Angular application*
Here we decided to use Heroku's fully-managed platform. The reason being that Heroku has a strong foundation and is run by a reputable software company. The driving motivation for using Heroku was the low cost thanks to the free tier offering.

**Server container**
*Spring Boot Application consisting of REST API routes*
For running our server, we decided to use the same Heroku setup as we did for the client container. The same benefits applied here and Heroku supported Java Maven applications.

**Database Container**
*MongoDB Atlas platform*
For the database we chose to use MongoDB Atlas because of its free tier and extensive platform. MongoDB Atlas comes with several tools for building aggregations within the application.

**User Device**
*Standard browser*
The application will be built using web technologies so this would require a browser to compile and run. The application is compatible with all browsers and is therefore accessible on all devices with a browser.

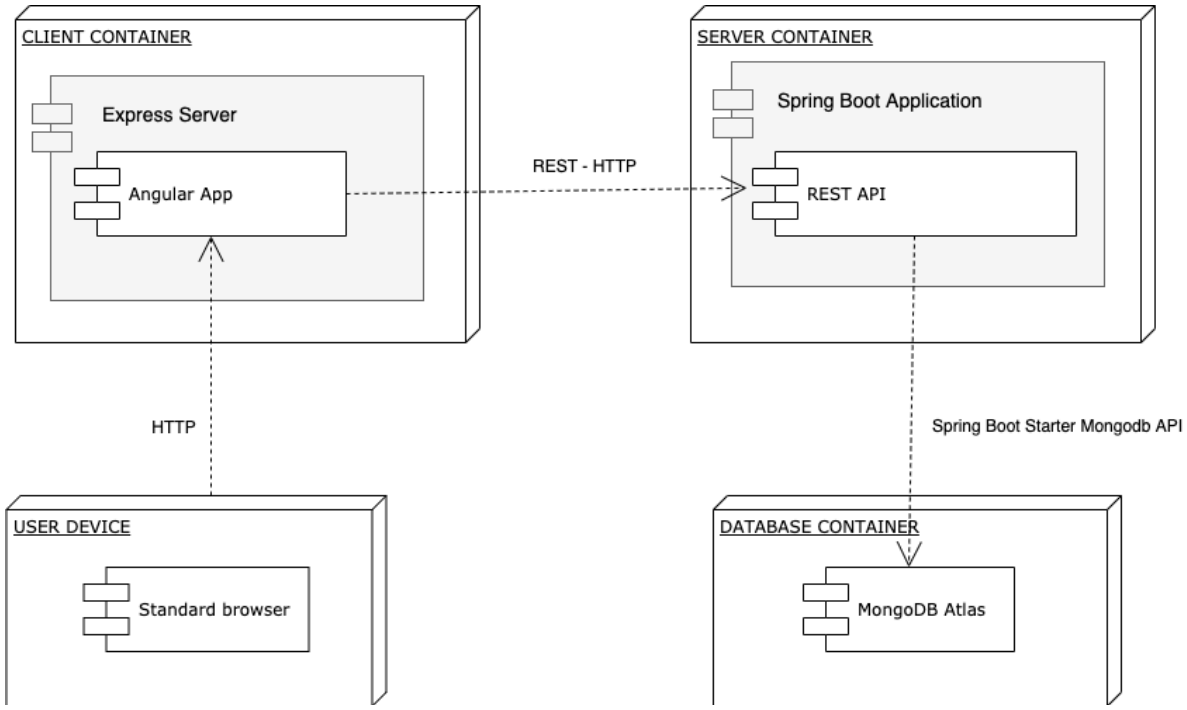



**Figure 4: A Deployment Diagram representing the dashboard**

# 7 ANALYTICAL REFLECTION

Looking back on the development of the sustainability dashboard we are, as a team able to draw some important conclusions. First of all, it should be noted that the application has been favored by the stakeholders of project EWA. There are multiple reasons for this:

**Our use of a uniform and simple design system has driven the time cost down for design and allowed us to build reliable and responsive screens. However, we did not utilize Angular as much as we wanted.**
The use of a simple design system has allowed us to create reliable screens fast. This allowed us to do quick prototyping and therefore push out more functionality. The screens we use throughout the application are similar in structure and allows us to cross develop. The design system is built on Bootstrap and that means we have enough documentation. We do think that we could have utilized Angular better. We had the possibility of using packages like Material Design and other design systems. We were unfortunately not able to utilize these packages because a lack of knowledge.

**The file structure is very understandable, divided and simple. The components within our application have clearly defined responsibilities and make optimal use of inheritance. We were not able make services store global state.**
The setup for our application directory was a priority. We have seen in previous projects that a cluttered directory has negative consequences down the line and often results in stalled development. We have tried to keep our directory low-depth and logical. We kept the creation of Angular components minimal. We evaded the use of boilerplate component creation through the *ng* commands. This creates 4 unnecessary files so instead we used inheritance to achieve a much more lightweight structure. The services we built did unfortunately not contain or cache data so we lack a global state in the application as described in *Alternatives*.

**The simplicity in our database structure has allowed us to implement functionalities quickly. The usage of NoSQL did make our aggregations large and complex to read.**
The statistics we gather are built up from documents and these do not require a relationship which allows us to be creative with aggregations. The lack of restrictions does make the implementation of new functionalities easier but the tradeoff is to be found in complexity. Our aggregations are very large and this is a point of improvement.