Sustainability

Architecture Document

Authors:
Aine Dwane *(s3445178)*
Rayyan Jafri *(s3687465)*
Hwanjun Lee *(s3543609)*
Erwin Nieuwlaar *(s2770873)*
Dante Santoso *(s3828557)*

Teaching Assistant:
Max Verbeek
Client:
Erik Hesselink

Contents

1 Introduction

VVE Sustainability is a project proposed by Erik Hesselink. The scope of this project is a web application to help VVE boards and apartment owners make sustainable and financially sound decisions by enabling communication between each other. The web application will help in that by being a source of information on the sustainable topics; facilitating messaging between VVE boards to each other, their members, and suppliers; and displaying the current steps and progress of each VVE in a clickable map. The requirement of map viewing is similar to Tesla's web page map.

2 Front end

Our client wants the platform to be very simple, compact, and clean. To achieve that Erik requested that it should have a flat structure, i.e. that each web page is accessible with as few clicks as possible. In our most recent meeting our client was able to give us the full details of his aesthetic specifications for the front end of the website, this is shown in image 1. An aesthetic design that our client wanted was a map similar to the one on Tesla's website. Instead of showing available dealerships, the map would instead contain locations of registered VVE's and their status on the seven sustainability topics. Furthermore, our client also expressed great interest in a three barred navigation button, the burger button, which would help maintain a flat structure for the website and permit a menu. For the maps we will be using the google maps API.

Figure 1: Homepage

3 Architectural Overview

3.1 General

Javascript, Node.js, and Express web framework interact with our MongoDB, noSQL database, for the reading and writing of user data such as Names, Associated VVE and Status within the VVE, messaging data such as All related chats and their messages and geopoint locations of VVE's to allow Google Maps API to query for this data and present a informative map on the front-end. Additionally, to convert the client's desires into a clear architecture and necessary functionalities a prototype was made in Axure, the subsection below is committed to this prototype.

3.2 Prototype

In the prototype all of the client's desires are processed, including some functionalities outside the scope of this project. In figure 2 a snapshot is shown of the designed prototype.

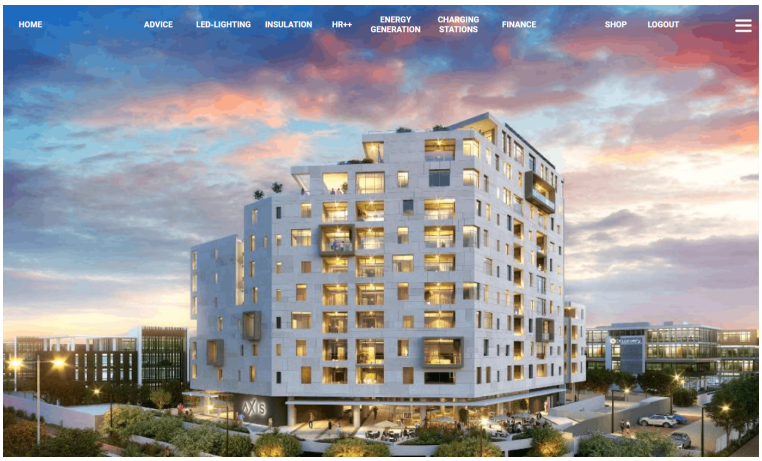


Figure 2: VVE Platform Prototype

Such a snapshot of a prototype may look nice, but does not reveal much architecture without context. To get to the right architecture, a lengthy call

was performed to get every user story into the prototype. According to these user stories a functionality graph was made in figure 4 to give the significant context to the prototype.

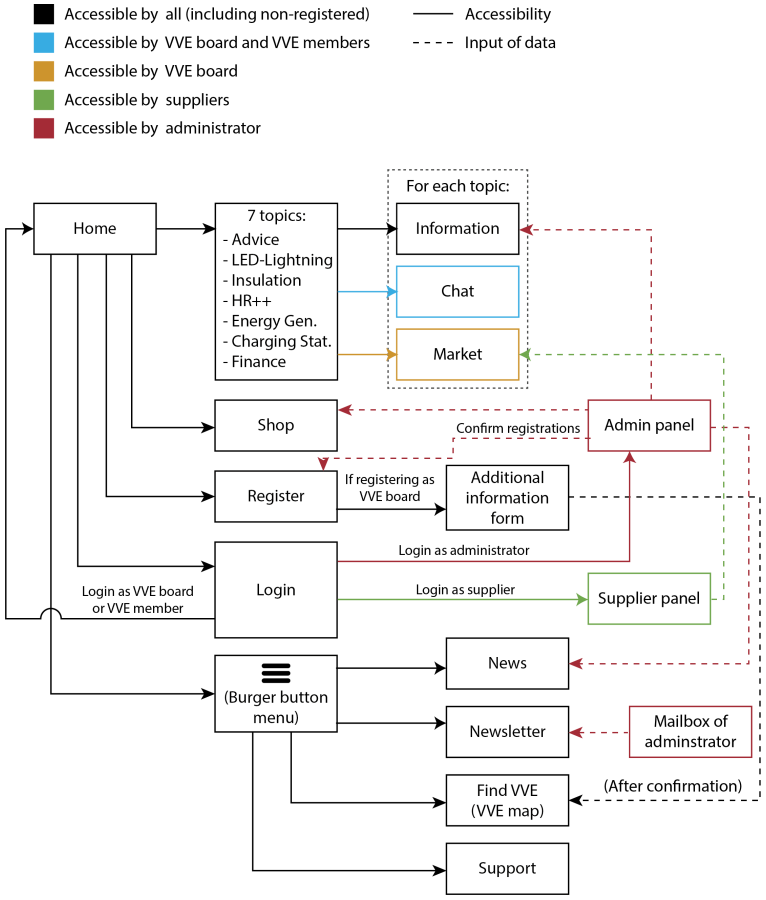


Figure 3: Functionality Diagram

In this diagram you can see the different degrees of accessibility of the users and where the input of data is supposed to come from. The boxes identify the pages and its functionalities. The accessibility of the VVE map is not completely correct represented in the functionality diagram, because the non-logged users the map should be visible but only the location and name of the VVE. For the VVE members, board and suppliers, the map should also include the stages of each sustainable topic. For the newsletter page (where you can subscribe to the newspaper), the newspapers are supposed to be generated from an email adress owned by Erik, the list of subscribed email addresses should be present in the admin panel. The input of the VVE map should come from the registration of VVE board members. When the VVE board member wants to register to the

platform and identifies himself as a board member, an extra information form will show with the needed information to register an VVE. After confirmation of the registration, the VVE should be added to the VVE map.

3.3 Authentication

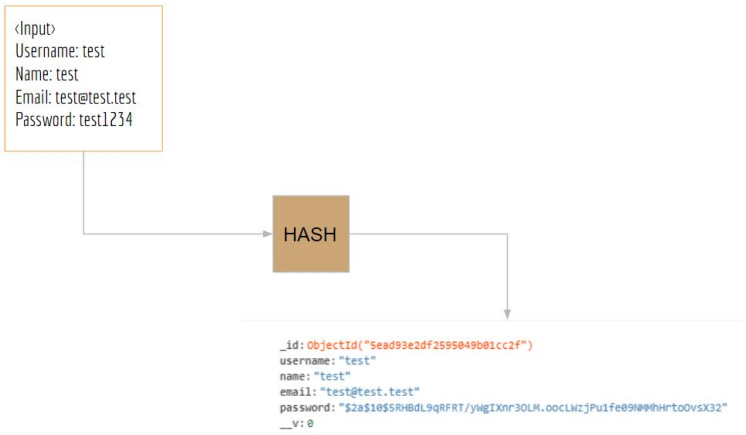
As we have not been able to find a way to automatically verify someone registering for the first time as a member of a VVE or a VVE Board member, for now they will need an invitation code. This works well for the early release of the product as the client would like to test the site with VVE's he is already connected through from his work in consultancy. Future development for vve board pages is automatic generation of invitation codes for their apartment owners.

3.3.1 Passport Middle-ware

PASSPORT is a middleware related to certification in NODE.JS. This is to perform login function or any function, and to access a specific page. It is to intercept and check the authority so that it is possible only when it is logged in order to access a specific page. In a routing environment, if you try to access a specific page, you have the ability to check permissions when accessing it by someone who is not external or authenticated. PASSPORT basically has a pattern of strategy to authenticate. The basic strategy is a local strategy in which developers can implement authentication logic directly using the database currently being serviced, and other strategies include using social APIs such as Facebook and Google, respectively. There is a library that provides these social strategies from each company, and it is flexible to change the strategy part to PASSPORT. The first time a user logs in or calls a specific URL, he or she is (isAuthenticated) and if he or she is not authorized or logged in, he or she is on the authentication module. Enter midware managed by passport to perform logic according to selected strategy.

3.4 Password Bcrypt

When the passwords are saved into the database, the passwords should be hashed for security reasons. For this reason, we use the password hashing function 'bcrypt' package to hash the password. For example, the user registers with below information. After hash, the password in the database is changed like following:



The code use the `compoundSync` function of bcrypt to verify that the stored hash matches the entered password hash. Also, in case there is a change in user.password during user creation or user modification, replace password with hash value with `bcrypt.hashSync` function.

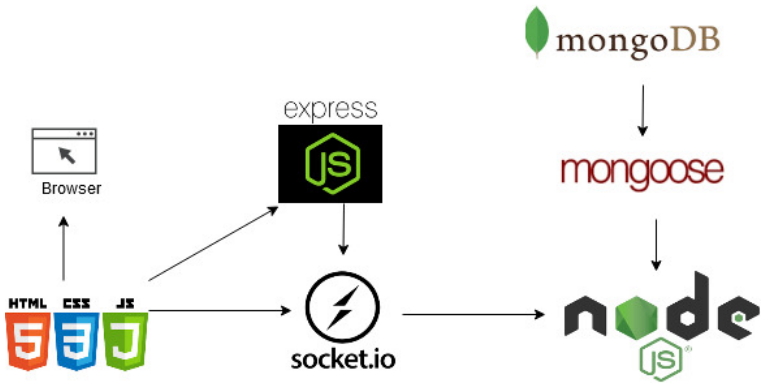


Figure 4: Technology Stack Relation

4 Technology Stack

4.1 Bootstrap

Bootstrap is a front-end framework for dynamic website and web application development, providing input windows, buttons, navigation and other components, and various layout as HTML and CSS based design templates, including additional JavaScript extensions.

4.2 JavaScript

Our daily website consists of three main components. They are "HTML(Hyper Text Markup Language)", "CSS(Cascading Style Sheets" and "JavaScript". HTML provides a large framework for web pages, and CSS manages design elements such as colors and handwriting. JavaScript is responsible for the operation of web pages in cross-platform, object-oriented script language. Our client want the website to be very simple and clean. To achieve that he requested that it have a flat structure; that each web page is accessible with as few clicks as possible. The style for this will again be very similar to the Tesla website.

4.3 Node.JS

Node JS is an event-handling I/O framework that operates on JavaScript engine 'V8'. It helps create applications with JavaScript in a server environment. The advantage of node JS is asynchronous programming. Sync programming means that you receive results immediately when you request something. The asynchronous request for an event does not require immediate result. This allows more diverse requests to be processed without waiting for results. In the web field, asynchronous programming was rarely used, but node JS made it relatively easy to program async.

4.4 Socket.io

Node.JS was used along with socket.io to implement chat functionality. Socket.io is a Javascript library that enables bi-directional communication between web clients and servers, thus making it perfect to be used to create a chat window.

4.5 MongoDB

MongoDB is typical NoSQL database system. MongoDB is easy to store and expand across multiple servers by storing data in the form of a Binary JSON document when exchanging data, and has the advantage of being fast in processing vast amounts of data due to its shallow depth structure.

4.5.1 Why using MongoDB

MongoDB, a noSQL database type was used for the back-end of the project. We choose noSQL since the data model itself is designed independently, making it easy to distribute data and to process data in an array at high speeds. This was largely chosen due to the clients demands for a real-time chat and large-scale user database which would require scalability and fast transaction queries. It is also a document-oriented database that fits the situation in which the schema must be changed frequently.

As our client wanted a real-time chat, this requires us to have a storage and retrieval method that is fast and can be updated quickly in real time which is what MongoDB can provide.

4.6 Express

Express JS (Express JS) is a web framework that allows developers to easily handle http requests. As with .Net in C# and Spring in Java, Express is used a lot in node.js.

4.6.1 Why using Express

The concept of EXPRESS is a web development framework that works on NODE.JS. Framework refers to an environment that provides standards for development and integrates and conveniently provides various other services. It has the advantage of being able to construct a web framework light and flexible. It also provides routing, static file hosting management, template engine, security, and other APIs to function as middle-ware for NODE.JS.

4.7 Interaction of the Stack Components

Beginning from the server-side; MongoDB lays down the foundational schema for data storage of the web application. In order to make server-side exchanges with MongoDB, node.JS is equipped with a mongoose module. Mongoose essentially allows the extraction/insertion of Data Models according to the schema

and protocol requirements of MongoDB. Through Mongoose, the required data is now available in Program Variables of Node.JS. Up until this point, our information exchange has remained on the server-side. In order for us to convey this data to the client side for front-end projection, 'Middle-ware' is required. Passport.js is authentication middleware for Node.js. As previously mentioned, we utilise passport.js as our middleware to interact with the client's HTTP requests. Infront of the middleware comes our client-side Javascript and Angular/HTML/EJS.

4.7.1 HTTP Protocol

This is in the form of a (request, response) cycle. When a user requests, for example, the home page. An Instance of Google Maps is loaded via Javascript on the client-side. A REQUEST is sent via HTTP protocols to our server, which then uses Express Node.JS Mongoose Extract Map Geolocations. The server script then pushes this data back to the client-side via Express and the RESPONSE of the HTTP request. I parse all data as JSON, since noSQL works on fundamental JSON and it is conveniently packaged as a JavaScript Object manipulating it at every stage of the server-client cycle is globalized/made easier.

4.7.2 Socket.IO Protocols

Socket.IO was an interesting element to implement as it's protocols required some more in-depth knowledge. The Server opens a WebSocket that latches onto any incoming requests routed towards our node server, in our instance, ip:8000. This required adjusting Socket.IO's protocol to send the websocket of type 'polling' on initial request and setting upgrade to false (Next section for details). This allowed the socket to connect faster and instantaneously as a WebSocket, instead of going through XHR/JSONP.
The socket is configured to the namespace and contains multiple chatrooms.

1. Server creates a 'nameSpace' for each chatroom
2. Socket opens for 'chats/chatroomName', all socket handshakes are handled by the nameSpace
3. Event listeners are attached to each nameSpace dynamically
4. Case: User connected:

4.7.3 Server-side Protocol

(a) Handshake protocol attaches a query "chatRoomID=user="

(b) chatroomID is compared with a list of current Mongo Models, only if it exists do we continue

(c) On the client side, if the username variable has not been instantianted this implies that the user is not logged in and incorrectly accessed this webpage, in which case we redirect the client to a 404 page.

(d) Assign the client socket to nameSpace: chatRoomID, assign the username : user

5. Case: On incoming message
   (a) The incoming message packet is routed to the relevant nameSpace socket
   (b) The type-tag of the packet is identified as 'chat message'
   (c) The nameSpace repacks this message with type-tag 'received'
   (d) nameSpace broadcasts the new packet to all listeners on its socket
6. Case: User disconnection
   (a) The incoming message packet is routed to the relevant nameSpace socket

4.7.4 Client-side Protocol

1. User accesses a route /chat/-chatroomName-
2. The server authenticates that the client is in a valid, logged in session
3. If client username is invalid, the socket will not be opened.
4. Socket connection is attempted on chatroom by a handshake
   (a) Create a packet with query parameters : chatRoomID=user=
   (b) Attempt a handshake with the server
   (c) Client receives socket object from server
   (d) Event listeners are attached to the socket objects
   (e) Case: Incoming Message
      i. Packet tag-type is identified as 'chat message'
      ii. Insert new message element into the HTML layout
   (f) Case: Outgoing Message
      i. Event listener is attached to the Send/Enter key
      ii. The page is prevented from refreshing (which is the natural feature)
      iii. The contents of the message box are packed into a message packet
      iv. Message Packet type-tag is assigned to 'chat message'
      v. Message Packet is given additional data: username, content, chatroom
      vi. Packet is pushed to the socket object for transfer

4.7.5 Web Server Configurations

Apache was used to construct the webserver on an Ubuntu 16.4 Linux System using remote shell. In order to allow the collaboration of Apache and Node, I had to configure Apache to run a ProxyPass on incoming requests via port *:80, routing them into localhost:8000 (Where the node server is listening).
In order to allow Socket.IO to work with the server, some deeper level of configurations had to be done. As mentioned earlier, Socket.IO initially sends an XHR/JSONP request with a possibility to upgrade to a WebSocket, the server then had to be configured to parse these HTTP headers as 'upgraded' and also add a parameter to the initialization of the socket, passing a socket of type 'polling' into the server from the client.

4.8 SonarQube

SonarQube is an application which detects flaws in coding and reports these shortcomings classified in four groups, namely; bugs, vulnerability, maintainability and security. Several SonarQube runs were performed on multiple branches, some bug fixing and refactoring was performed according to the findings of SonarQube. However, SonarQube was not a game changer in our project, as most of the time, it returned minor warnings. At the end of the project we had SonarQube running over our final master branch, the report is shown in figure 5.

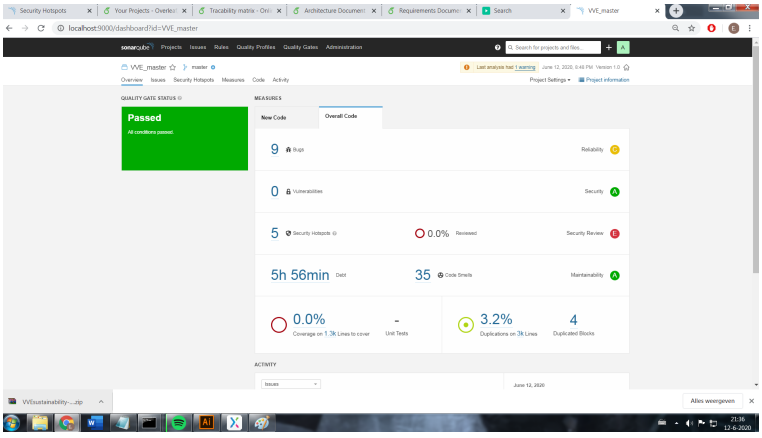


Figure 5: SonarQube Report Pre-final

Additionally, SonarQube provides indepth explaination of the found issues, the list of bugs found from the report in figure 5 is shown in figure 6.

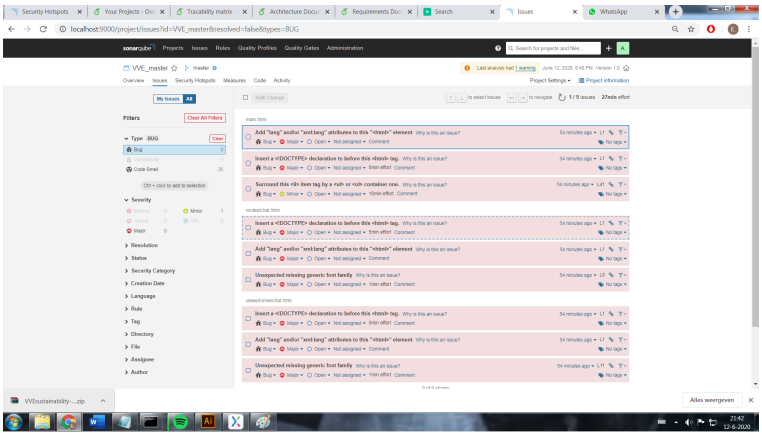


Figure 6: SonarQube Bugs list

Thanks to the cooperation of our team we managed to solve most of the last issues addressed in figure 5 and 6, just before the deadline (see timestamps of screenshots). The final SonarQube report is shown in figure 7.

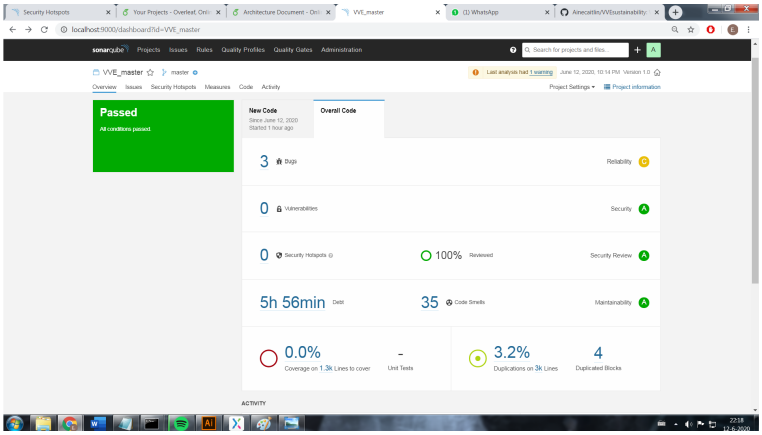
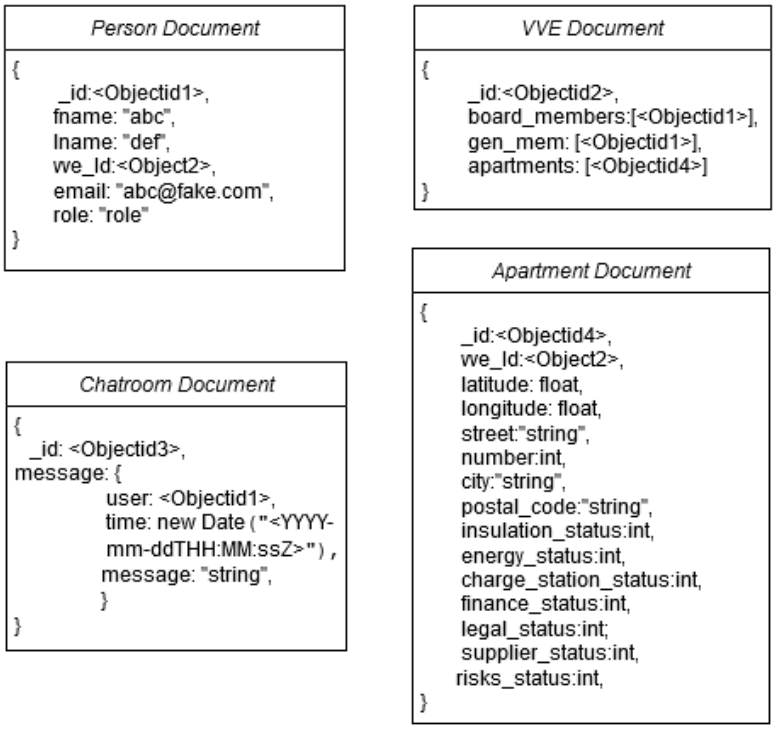


Figure 7: SonarQube Report Final

5 Database



The scalability factor of a non-relational schema draws its benefits from a shallow structured database. To fully utilise this in terms of efficiency of queries and reducing client transaction costs; multiple instances of the same data are embedded under objects. This allows fast, precise transaction times that aren't a burden on bandwidth. An example can be Board Members embedded into the VVE Document, so a query for a specific VVE will have the relevant object embedded instead of providing relations to another table where multiple key-matching queries through a table would have taken place to find the associated board members. This implementation works best for a one-to-few type relation of documents. Only logged in users may access the chat features.
To further elaborate on the 'Chatroom' Document, every chatroom will have its own Document i.e 'General-Chat' with a large amount of messages [sender, content, time] associated with it that have to be retrieved in correct order. There was also the issue of memory usage in terms of efficiency, so MongoDB's capped collection feature was used to make faster indexing on retrieval of this special document coupled with removing insertions from oldest to newest whenever the memory cap is exceed: 524,288,000 Bytes was found to be a suitable memory allocation for each Chat.

Instead of opting for embedding these as one would with a one-to-few type relation, we instead will create a parent reference from 'message' to their respective parent 'Chatroom'. This will provide ease of use when a user requires the messages of a specific chatroom, in our application, this is a high frequency query as chatrooms are for specific VVE's thus a lack of a global chatroom. By implementing this, we can boil down our chatroom query to a highly precise pull of data with a query :

- $db.categories.find(parent : "Chatroom1337")$

The creation of new chatrooms is completely dynamic, the admin simply needs to input a new name for the chatroom. All sockets, routing and database creation is implemented in the code.

6 Team Organization

We have decided to implement scrum more over time. At the beginning of the project and due to the corona virus outbreak the scrum method had been left behind a bit. However, in the final stage of the first block and the second half of the project the intention is to make fully use of the Scrum method. We use Trello to keep track of our tasks and responsibilities and slack for communications.

In the later stages of the second block, it was decided by some members to implement a Scrum master for roughly 1 week periods per person. This was done as at that point in time, while each of us have something to add to the final product. There was some confusion on how to progress further and how to fit everything together.

This decision in implementing a scrum master was extremely useful in advancing our progress and making sure everyone knew what to do and how each of everyone's projects fitted together.

6.1 Front End Team

6.1.1 Hwajun

1. Making front end page based on prototype

6.1.2 Rayyan

1. Chat System front end
2. Google Maps Display

6.1.3 Dante

1. Chat System front end

6.2 Back End Team

6.2.1 Rayyan

1. Design and Implementation of Database Schema
2. Chat System Back-end
3. Google Maps API
4. Bug Fixing Login System

6.2.2 Hwajun

1. Login System
2. Basic Register System
3. Search (Board) Page

6.2.3 Dante

1. Chat System Back-end

6.2.4 Aine

1. Role based Registration Page

6.3 System Administration

6.3.1 Rayyan

1. Setup Remote Server
2. Configure Apache Web Server, FTP Server, Node.JS Server, Socket.IO configurations
3. User Assignemnt, Proxy,Routing, File Permissions

7 Change Log

Who	When	Which Section	What
Aine	28-3-2020	Introduction	add background information
Hwajun	1-4-2020	Technology Stack	make framework
Aine, Erwin, Dante, Hwajun	2-4-2020	Whole section	add technology stack and front end
Rayyan	2-4-2020	Architectural Overview	
Rayyan	6-4-2020	Database	Paragraphs + Model Description + Code Snippet
Aine, Hwajun	5-4-2020	Introduction and Architectural Overview	add Auth0
Aine	5-4-2020	add model and model description	
Hwajun	3-5-2020	Technology Stack	add express
Hwajun	27-5-2020	Technology Stack	Architecture overview and Technology Stack
Hwajun	04-6-2020	Architecture overview	add bcrypt and passport middle-ware
Rayyan	05-6-2020	Entire Interaction of the Stack Components Section	
Rayyan	05-6-2020	HTTP Protocols	
Rayyan	05-6-2020	Apache Configurations	
Rayyan	05-6-2020	Socket.IO Protocols	
Dante	05-06-2020	Why using MongoDB	Added a small explanation
Dante	05-06-2020	Team organization	Added scrum master decision
Rayyan	12-06-2020	Added use cases for Socket.IO Handshake Protocol	
Erwin	12-06-2020	All	Reviewed document, added prototype, functionality diagram and SonarQube