Paper submitted within the scope of the Bachelor's Project
Bachelor of industrial Sciences
KU LEUVEN - Faculty of Engineering Technology, Campus Group T Leuven

Title

Cemal Yetismis
KU LEUVEN - Faculty of Engineering Technology, Campus Group T Leuven

Coach: Jeroen Van Aken;
KU LEUVEN - Faculty of Engineering Technology, Campus Group T Leuven,
jeroen.vanaken@kuleuven.be

1 Introduction

The real-life problem that was given by the customer is reading and visualizing temperatures through sensors in current time measurements as well as past measurements. In addition to these features this a web-based application had to be user friendly and easy to comprehend, so introduce the web-based app with some additional features such that, each user can access their account privately by scanning the QR code of the website, user can visualize each sensors temperatures individually via line charts and heatmap, by logging in as an admin the user has some privileges, for instance, the admin add or remove the sensor. While developing this site I focused on how a user can visualize, manipulate the data better and faster during the fire training session.

2 Requirements & background

I. This monitoring system will be used by fire-fighters during their training sessions. They want to visualize the behavior of the fire so that they can show a reaction to the danger of the fire and they can find new defense tactics against the expansion of the fire.

II. The temperatures data which are obtained from ten various sensors that are located different places in the shipping container, need to be visualized at most 30 seconds intervals.

III. The temperature data need to be monitored via the web-based application that is compatible with all devices (tablet, smartphone, computer, etc.).

IV. During the training session, the firefighters don't want to waste their time to receive data since they are in a hurry and great danger at that moment. Thus, they expect to obtain the data instantly in the most efficient way and with as least click as possible.

V. There have to be different types of visualizing tools such that charts or maps, to analyze the data received before or throughout the training session.

VI. There will be one shipping container and the web-based app needs to show the sensor's values in this container. The developer has to consider this condition when developing the site.

VII. The instructors want to reach the website via scanning the QR code.

VIII. The website page has to fast respond to the Http requests of the fire-fighters. There has to be no overload on the pages.

IX. The web site should be neat and does not include redundant data during the training session. For instance, you are at the training session and you just need the data which includes only the last 30 second intervals.

X. Also, after the session, the users have to reach the past data easily such that in the web page screen by entering the time and the session id, the user has possessed all of the data related to those entered features.

3 Design & Implementation

This chapter extensively focuses on how the web-based app was architecturally designed and which technologies were used as front end, back end, and database.

3.1 Top Level Design

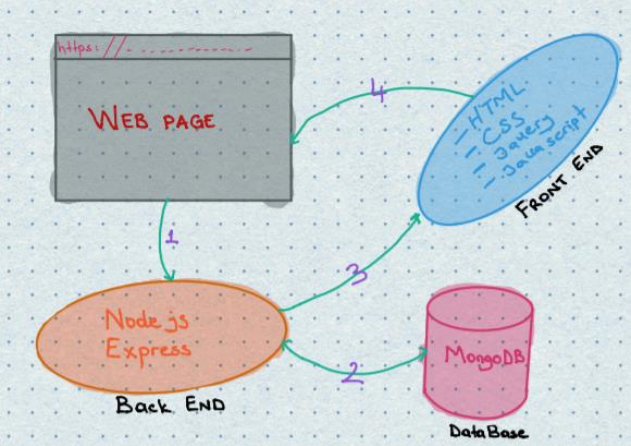

Figure 1: TLD of Website[1]

As can be seen from Figure 1 in order to develop the website are needed four main parts. The important part of this figure is the database section which sensors and user's data are stored in MongoDB as schema. The back-end technologies (Node.js and Express were chosen for this website). was used to retrieve these data and send it to the front-end section. To display the obtained data from the back-end as a well-structured and user-readable manner, the front-end technologies HTML5, CSS, J-query, and Java-script were used. The entire process begins with 'https' request from the user, are shown in Figure-2.

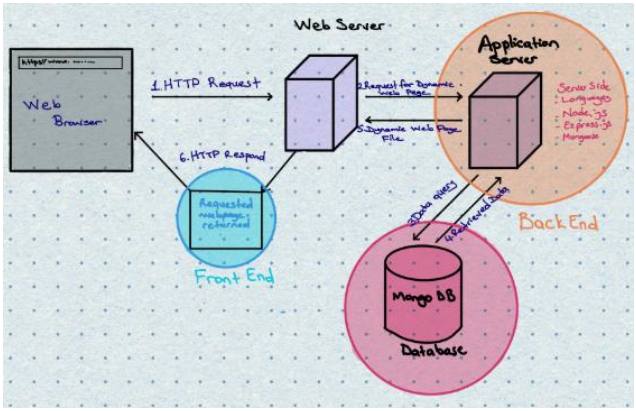

Figure 2: The Web side process to monitor the sensor's data[2]

3.2 Design Choices

According to the project's aim, the website was constructed as a dynamic web-page structure. Because every 30 seconds the sensors data need to be shown on the web-page. There are a lot of back-end technologies for developing a website. For this project node.js and express.js are chosen for the back-end part due to their advantages compared to other techonologies.Node.js enables developers to develop front-end and back-end parts of a web-site in one language. Moreover, with asynchronous non-blocking input/output features of node.js the website can respond to hundreds of thousands of simultaneous requests. The developers benefit from different collection modules of node.js NPM package manager as it was used for the authentication part of the firefighter web-site. In addition to these features, node.js offers express.js library to develop efficient, fast, and scalable web-app and to shape your server-side MVC implementation efficiently.

In point of database, MongoDB was chosen as database technology for this project. Since I have more experience as a developer with MongoDB and there was no certain database technology preference of customers. Additionally, there is a close-link between MongoDB and Node.js. Document in MongoDB is stored in BSON files which are, in fact, a little-modified version of JSON files. Because of this, it is frequently used for Node.js projects. Moreover, it

[1] Alex Coleman, High Level Design of a Web Application, Retrieved 15 May 2020 from https://selftaughtcoders.com/from-idea-to-launch/lesson-12/high-level-design-web-application-controllers-views/

[2] https://developer.mozilla.org, Client-Server Overview, Retrieved 15 May 2020 from https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview

offers greater efficiency and reliability which in turn can meet your storage capacity and speed demands.

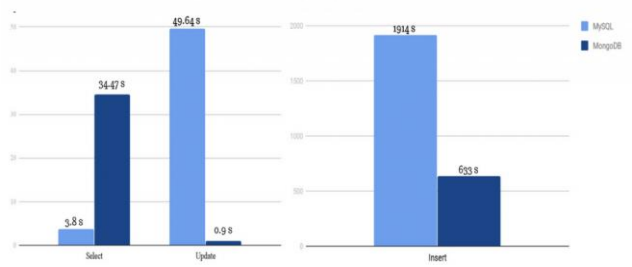

Figure 3: MongoDB vs MySQL query speed performance (https://www.simform.com/)[3]

It is evident from figure 3 that MongoDB takes a much shorter time than MySQL for the same commands except the select statement.

MongoDB also offers great performance analysis tools like the graph for database metrics measurements. Thereby you can do measurements in an efficient and user-friendly way.

This website has a usage for firefighter training so it needs to be opened to improve in many ways such as they can upload training session videos and images in the future. Thus, MongoDB is more suitable for this kind of usage than MySQL.

3.3 Development

3.3.1 MVC Implementation

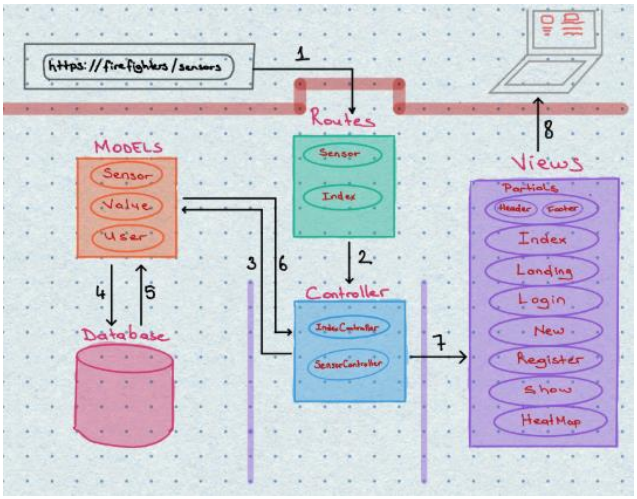

Figure 4: MVC implementation more detailed[4]

This section focuses on the Model View Controller implementation on the web-based monitoring tool. As we can see the Figure-4, MVC implementation of the web-site consists of five main parts. The entire process begins the 'https' request as it was mentioned in Figure-2. However, Figure-4 shows the process in detail in terms of MVC implementation. Moreover, it points out the order of the process. The process includes 8 different steps. If it is looked broad perspective at the process, after a firefighter leader entered URL by scanning the QR code, it is matched to a route that associates the URL with a designated controller action. Then the controller leverages the necessary models — which interact with the database — to retrieve the necessary information. It then sends that information off to a view, which renders the request page for the user to view in their browser. Finally the leader can monitor the desired web-page.

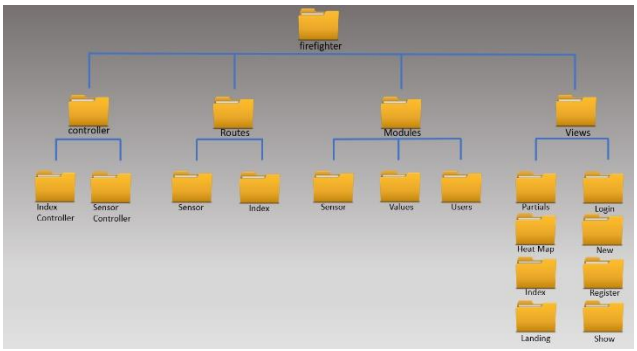

Figure 5: MVC implementation the web-site in as file tree

3.3.1.1 Routes

Routes are the first phase in the flow diagram and it is responsible for directing desired URLs to the controller. As can be seen from Figure 4 and Figure 5 the website routes include 2 subfolders, sensor and Index. The routes are designed as CreateReadUpdateDelete structure.

Routes	Explanation
/sensor	direct to all sensor
/new	direct to add a new sensor

[3] Mihir Shah, Retrieved 15 May from https://www.simform.com/mongodb-vs-mysql-databases/

[4] Alex Coleman, High Level Design of an MVC Web Application: Using Models to Interact With a Database, Retrieved 15 May 2020 from https://selftaughtcoders.com/from-idea-to-launch/lesson-18/high-level-design-mvc-web-application-using-models-to-interact-with-database/

/sensor/:id	direct to more info about sensor

Table 1: Sensor routes

Routes	Explanation
/	direct to the landing page
/login	direct to login page
/register	direct to register page
/heatmap	direct to heatmap page
/about	direct to about page
/logout	handling to logout

Table 2: Index routes

3.3.1.2 Controllers

As can be seen from Figure 4 the controller is the second phase in the flow diagram and its mission is to send required queries to the database and retrieve the desired data. It consists of 2 child nodes SensorController and IndexController. The task of each function for each child node is explained below.

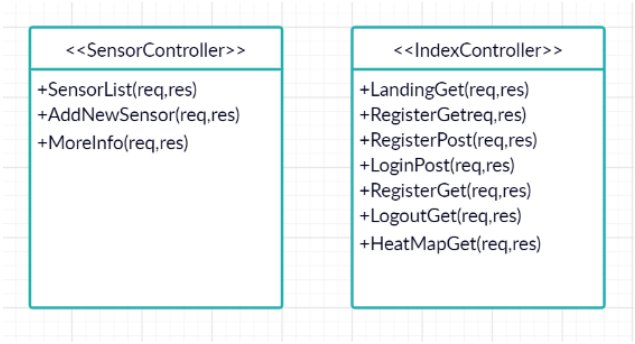

Figure 6: Functions in Sensor and Index Controllers

SensorList(req, res): It sends the query to the database to obtain all sensors with last temperature value, location, and sensorID. It populates the last value from the database. Since the query populates 1 last value rather than populating all temperature values, it protects the database from overloading redundant data. As a result,

he sensor page is loading much faster.

AddNewSensor(res,req): This function handles process adding new sensor to database.

MoreInfo(req, res): It helps to retrieve the last 20 temperatures value and time value to draw a graph from the corresponding sensor.

LandingGet(req, res): It brings the home page of the website.

RegisterGet(req, res): It registers the user who filled in the required information, after the post request.

RegisterPost(req, res): This function send to post request to insert user to the database.

LoginPost(req, res): It sends and checks filled user information to the database to carry out login activity.

LoginGet(req, res): It brings the login page.

LogoutGet(req, res): It performs the Logout process.

HeatMapGet(req, res): It obtains the last 10 sensor temperature data from the database and renders it to the heatmap page.

3.3.1.3 Models

DATABASE SIZE: 2.78MB INDEX SIZE: 2.34MB TOTAL COLLECTIONS: 3

Collection Name	Documents	Documents Size	Documents Avg
sensors	10	1.01MB	103.82KB
users	1	1.14KB	1.14KB
values	26417	1.76MB	70B

Figure 7 Database storage for 26417 temperatures values[5]

Indexes	Index Size	Index Avg
2	1.14MB	586KB
1	36KB	36KB
2	1.16MB	592KB

Figure 8 Database index[6] *storages*

[5] cloud.mongodb.com,Collections Retrieved 15 May 2020 from https://cloud.mongodb.com/v2/5ea35f8f0b75ca4ba24f3442#metrics/replicaSet/5ea3601fabd5e7733b4d61dc/explorer

[6] cloud.mongodb.com,Collections Retrieved 15 May 2020 from https://cloud.mongodb.com/v2/5ea35f8f0b75ca4ba24f3442#metrics/replicaSet/5ea3601fabd5e7733b4d61dc/explorer

The model is the third phase as it is seen in Figure 4 and it contains the MongoDB schemas[7]. The database architecture is shaped according to the application's aim and expectations. As regards to the best performance and usage requirement, the database of this web application is comprised of three different schemas such that Sensor schema, Value schema, and User Schema.

```
var sensorsSchema = new mongoose.Schema({
    sensorId : Number,
    location : String,
    values: [
        {
            type :mongoose.Schema.Types.ObjectId,
            ref :"Value"
        }
    ]

})
```

Figure 9 Sensor Schema

Sensor Schema: Sensor schema has three different fields as can be seen from Figure 9. Each sensor has to have sensorId to make each sensor unique. By making each sensor unique, it prevents errors and misunderstanding when the data is processed. The location field is added to the database schema to determine the location of the sensors in the shipping container. It is chosen string but it can be switched to number as a coordinate according to customer desire. In respect to temperature, the value field is added to sensor schema. As each sensor must have temperature value which is the main concern of the project. Therefore, the values field is added to sensor schema as an array and it refers[8] to Value schema.

```
var valueSchema = new mongoose.Schema({
    value       : Number,
    time        : Date,
    tsession    : Number,


});
```

Figure 10 Value Schema

Value Schema: Value schema is constituted of three particular fields, these are value, time, and tsession. The value field represents the temperature readings by one sensor. The time field is the current time for reading temperature value. Tsession's field represents the training sessions for one day. It is assumed 5 particular tsession will organize for each day, however, the amount of training session can be increased or decreased to fire fighter's expectation. Tsession helps to retrieve past temperature data when the firefighters want to analyze past pieces of training data. Moreover, each tsession has its temperature values.

```
var UserSchema = new mongoose.Schema({
    username: String,
    passwords : String,
    isAdmin :{type:Boolean, default: false}
});

UserSchema.plugin(passportLocalMongoose);

module.exports = mongoose.model("User",UserSchema);
```

Figure 11 User Schema

User Schema: User schema is designed to keep username and password as the most website does. The isAdmin field is put to the schema to give privilege to some users. This user can add a new sensor or can add a new user to the system. These privileges can be enhanced according to the wishes of firefighters.

[7] docs.mongodb, Document Structure, Retrieved 17 May 2020 from https://docs.mongodb.com/manual/core/data-modeling-introduction/

[8] docs.mongodb, References, Retrieved 17 May 2020 from https://docs.mongodb.com/manual/core/data-modeling-introduction/

There is a secret admin code if a user enters this code during the registration process this user will be admin and can use the admin's privilege.

3.3.1.3.1 Analyzing and Calculating Storage Needs

Assumption: This assumption made by taking into customer consideration.

Each sensor will create one temperature value every 30 seconds and it is 2 for every minute and 120 for an hour. Ten sensors will create a total of 1200 temperature values in one hour.

It is assumed that a day consists of 5 particular training session which takes 1,5 hours long.

According to the above calculation, ten sensors will create approximately;

daily 9.000,

weekly 45.000 (weekdays),

monthly 180.000

yearly 2.160.000 values

DATABASE SIZE: 2.78MB INDEX SIZE: 2.34MB TOTAL COLLECTIONS: 3

Collection Name	Documents	Documents Size	Documents Avg
sensors	10	1.01MB	103.82KB
users	1	1.14KB	1.14KB
values	26417	1.76MB	70B

Figure 12 Current Database Storage from source

According to Figure 12;

One temperature value document takes 70byte space in the memory

One user document takes 1.14Kb space in memory

One sensor document which has 26417 temperatures value, takes 1,76 MB space in memory

One temperature value effects sensor schema 17,92 byte since it contains the values field IDs. As a result one sensor space rises when one value field is added to the database.

To calculate storage needs is made use of current database storage from MongoDB Atlas.

2.160.000-26417=2.133.870 extra value is needed for current database,

2.133.870*70byte=145.850Kb=142,432 Mb is needed for temperatures value,

2.133.870*17,92byte=37.342,725=36,467Mb

Effect of temperatures on sensor schema,

Total 142,432+36,467=178.899Mb,

Current database storage =2.76 Mb

Value and Sensor storage =178,901 MB for database.

Indexes=190 Mb for indexes

According To 5.000.000 Dataset:

Figure 12 shows,

Current Database storage: 1.01Mb,

Current Temperature Value number: 12130,

As a result;

Desired dataset:5.000.000,

5.000.000-26417=4.973.583 extra data needed

4.973.870*70byte=332,022MB
is needed for temperatures value,

4.973.870*17,92byte=85 MB

Effect of temperatures on sensor schema,

332,022+85+2,74=419,762 Mb

Avg User Document:1,14Kb,

Index storage = 422 Mb

Total database storage needs for 5.000.000.000 dataset is **419,217 MB**.

Database cleaning can be arranged at a regular interval. In this way database storage can be reduced. and query speed will be gone up. The other option can be creating a second database for past-data. By building the second database the query speed of live data can be boosted since the queries search the desired value in less amount of data. However, in this calculation, it is not taken into consideration.

3.3.1.3.2 Performance metrics/optimizations for Database

When the website's database was first created, there were no indexes on the modules. Moreover, queries were written without limitation and specification. After the dataset of the website was grown there occurred much more delay while the sensor page was loading. To refrain from these delays there were a couple of optimizations implemented to the database and the queries to enhance database response.

```
var mongoose= require("mongoose");
var sensorsSchema = new mongoose.Schema({
    sensorId : Number,
    location : String,
    values: [
        {
            type :mongoose.Schema.Types.ObjectId,
            ref :"Value"

        }
    ]

})
 sensorsSchema.index({ sensorId: 1, values: -1 });
module.exports = mongoose.model("Sensor",sensorsSchema);
```

```
var mongoose = require("mongoose");

var valueSchema = new mongoose.Schema({
    value       : Number,
    time        : Date,
    tsession    : Number


});


valueSchema.index({ value: -1});
module.exports = mongoose.model("Value",valueSchema);
```

Figure 13 Current Sensor and Value Models

Firstly, after that indexes[9], as is seen from Figure-13, were put into models, it would take a shorter time to load the sensor page. Since the indexes increase to query efficiency. Moreover, when the indexes were created, some perquisites taken into consideration. They have must increase to the current query speed. So they were designed to this aim. However more indexes bring overload to database storage, it has to be in equilibrium. For this reason, it is refrained to add redundant indexes.

```
Sensor.find({})
    .populate({
        path: 'values',
        options: {
            sort: { _id: -1 },
            limit: 1
        }
    })
```

```
Sensor.findById(req.params.id)
    .populate({
        path: 'values',
        options: {
            sort: { _id: -1 },
            limit: 20
        }
    })
```

Figure 14 Optimized queries

Secondly, since values field refers to value schema and it was needed to populate to show temperature value, previously all values collection was populated to show temperature values on the sensor page. But to populate all values was redundant and it brought an extra load to network efficiency. To avoid this burden only required values were populated[10]. For instance, as can be seen from Figure-14, the last temperature value for each sensor and last 20 temperature and time value, for the sensor which is wished to analyze, were populated instead of all value collection.

As was mentioned previously second database can be built for past sensor data. This data can be transferred daily or weekly to the second database. The current database includes only daily data. In this way query speed will immensely rise.

3.3.1.4 Views

The view is the last phase before the user visualizes the web page. It has a direct impact on the loading speed of the web site. Furthermore, it needs to be designed efficiently. The used fonts, images, icons, and libraries such as bootstrap or chart.js has to be chosen carefully. Because the design of the view offers not only adequate speed but also has to be user friendly. The view includes 8 different view pages as can be seen in Figure-4 and Figure-5.

The first view folder is to be named as partials that consist of header and footer. These two '.ejs'[11] file is created to avoid writing the navbar, google font, chart.js, script tags, etc. to all files iteratively. Thus, with a simple 2 lines code, they were added to other view files.

---

[9] https://docs.mongodb.com, Indexes, Retrieved 17 May 2020 from https://docs.mongodb.com/manual/indexes/

[10] https://mongoosejs.com/, limit vs. perDocumentLimit Retrieved 17 May 2020 from https://mongoosejs.com/docs/populate.html

[11] https://ejs.co/#docs, Tags, Retrieved 18 May 2020 from https://ejs.co/#docs

The second view file is Index.ejs file which shows 10 sensors with last temperature value. One of the great advantages which node.js bring to the developer is that the javascript code can be added[12] directly to 'ejs' file. This powerful feature gives developers a great advantage. To illustrate, by using for each loop, 10 sensors can be shown with their fields such as location, temperature, and ID. During the designing of this page, the user expectations were taken into account. For instance, to monitor the last value of each sensor on one web-page, this page saves the user time during the training and gives accumulate views. If the user wishes to analyze further the sensor data 'more info' button is needed to click.

The third view page shows .ejs. After clicking the 'more info' button chosen sensor's graph can be seen as an apart page. In this page chart.js library was used to create easy on the eye graph.

The fourth view's page is landing.ejs. This page can be called a home page. The home pages show the design quality of your website. Furthermore, they have to leave the best impression on the users. They have to be attractive and also be loaded in a few mili seconds. Thus, to attract the user firefighter images were chosen as a landing page.

The fifth view's page is heatmap.ejs[13]. This page is laid out as a heatmap of the shipping container. To create a more efficient heatmap is needed more sensor.10 sensor is not adequate for the heatmap. However, the heatmap page is added to the web site and the customers will have the decision. If they want, the heatmap.ejs page can be removed from the project or it can be improved if more information can be taken about the expansion behavior of the fire in the shipping container.

The other view pages are login.ejs, register.ejs and new.ejs. Login.ejs and register.ejs handles the sign-in and sign-up process of the user. New.ejs is the layout for adding a new sensor.

3.3.1.4.1 Bandwidth Consumption and Performance Measurements for Main Pages

This measurement was performed with 26770 datasets.

Landing Page:

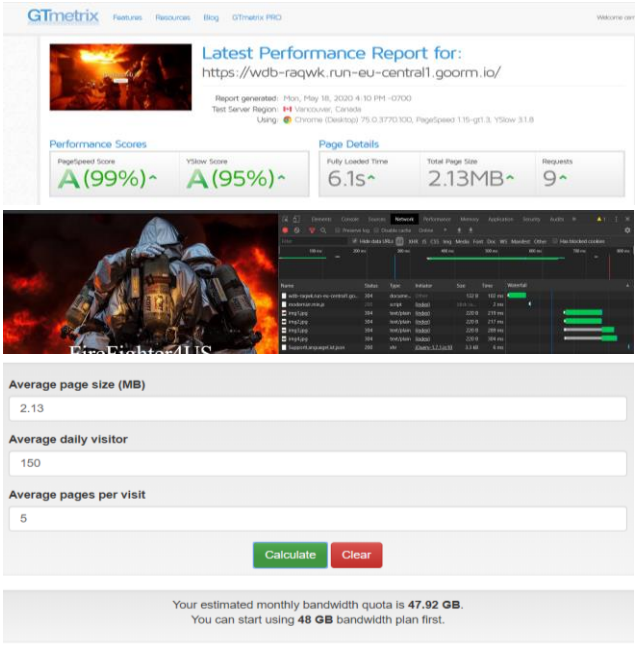

Figure 15 Performance and bandwidth measurement landing page[14]

To calculate bandwidth some assumptions are made for the average daily visitor and average pages per visit. To diminish bandwidth consumption, you can decrease the web page size. However, as we discussed in the previous section the view of the landing page has to be attractive. Moreover, the landing page will not be viewed during the training session since the QR code was created to direct users to the sensor page.

Index Page:

[12] https://ejs.co/#docs, Includes, Retrieved 18 May 2020 from https://ejs.co/#docs

[13] Patrick Wied, h337, Retrieved 19 May 2020 from https://www.patrick-wied.at/static/heatmapjs/docs.html

[14] https://gtmetrix.com/, Analyze,Retrieved 19 May 2020 from https://gtmetrix.com/

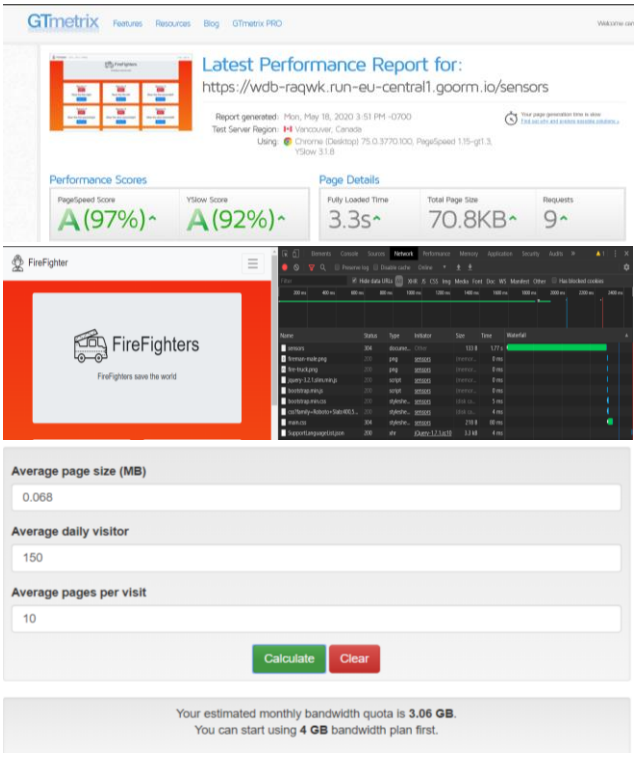

Figure 16 Performance and bandwidth measurement index page (https://gtmetrix.com/)[15]

Since this page is going to be used during the training session, the web page size kept minimal.

Show Page:

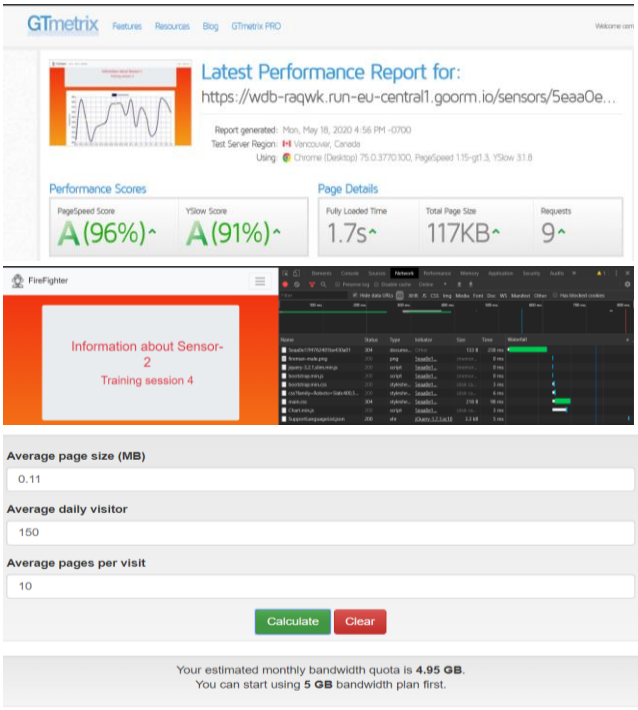

Figure 17 Performance and bandwidth measurements show page (https://gtmetrix.com/)[16]

Heatmap:

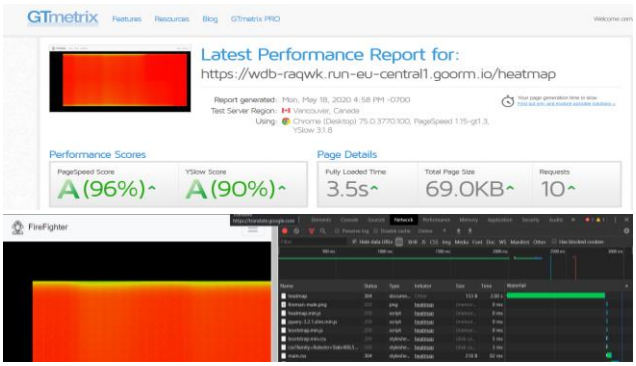

Figure 18 Performance measurements heatmap page (https://gtmetrix.com/)

3.3.1.4.2 Implemented Optimizations for Views

In order to load a page, the browser must parse the contents of all <script> tags, which adds additional time to the page load.[17] To diminish load time of the

[15] https://gtmetrix.com/, Analyze,Retrieved 19 May 2020 from https://gtmetrix.com/

[16] https://gtmetrix.com/, Page Speed ,Retrieved 19 May 2020 from https://gtmetrix.com/reports/wdb-raqwk.run-eu-central1.goorm.io/4izjqZ08

[17] https://gtmetrix.com/, Page Speed ,Retrieved 19 May 2020 from https://gtmetrix.com/reports/wdb-raqwk.run-eu-central1.goorm.io/owe8TINt

page <script> tags were used as less as possible in the footer and header partials.

- https://cdn.jsdelivr.net/npm/chart.js@2.9.3/dist/Chart.min.js (162.2KiB)
- https://code.jquery.com/jquery-3.2.1.slim.min.js (62.7KiB)
- https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js (42.4KiB)
- https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js (16.7KiB)
- https://cdn.jsdelivr.net/npm/heatmapjs@2.0.2/heatmap.min.js (8.2KiB)
- https://wdb-raqwk.run-eu-central1.goorm.io/sensors (70B of inline JavaScript)

Moreover, in order to perform this optimization, a minified version of the CSS<script> tags and JavaScript codes were used.[18]

The <script> tags in the header file are moved to the pages header in which they are used. Because there is no need to call chart.js for heatmap or index page.

The small image sizes are preferred except the landing page, in order to maintain the performance efficiency.

External CSS files and very small inline CSS codes are used to enhance performance.

Inline style blocks and <link> elements were put at the document head to improve rendering performance.

The compressed version of the images was used for the landing page.

The asynchronous resources are preferred.

The fast-loaded font type and size were chosen.

4 Testing

4.1 Profiling Code and Indicating Bottlenecks of the Project[19]

Figure 19 Test condition with clinic.js[20]

Figure 20 Profiling code bottlenecks with clinic.js

As can be seen from the figure above, firstly it takes pretty much time to provide connection, and secondly, since sensors create data 30 seconds this data brings high load to controllers that get desired data from the database. The more connection brings more requests to the database and as a result of that, this leads to congestion of the connection. The written code which gets all sensor with last data, search all sensor data and brings the last result. As we mentioned at the database section these queries were optimized however it adds still extra load. This problem can be evaluated as a bottleneck of the codes. The reason for the other latencies is that when this project was creating, the goorm.ide developer area as server and computer, MongoDB Atlas as a database platform used. These brought extra latencies to connections. Since in the real-life project it will be created a particular server and database for the project, the latencies time will be shorter. One notice that

[18] https://gtmetrix.com/, Page Speed ,Retrieved 19 May 2020 from https://gtmetrix.com/reports/wdb-raqwk.run-eu-central1.goorm.io/4izjqZ08

[19] Matteo Collina,A New Way To Profile Node.js, Retrieved 21 May from https://www.infoq.com/presentations/profile-nodejs/

[20] Matteo Collina,Finding Bottlenecks Retrieved 21 May from https://www.infoq.com/presentations/profile-nodejs/

MongoDB time is also high however this is related to test connection time it was adjusted 10 s time duration for this test. Moreover, in this project express-session was used in-memory store. This allocates and maintains a lot of memory (by design)[21]. That is the main reason for the latency.

Figure 21 Profiling code bottlenecks with clinic.js indexes are added

The figure shows that if indexes are added to the database the latency time goes up approximately two times than without indexes and there hasn't occurred any improvement at the other latencies.

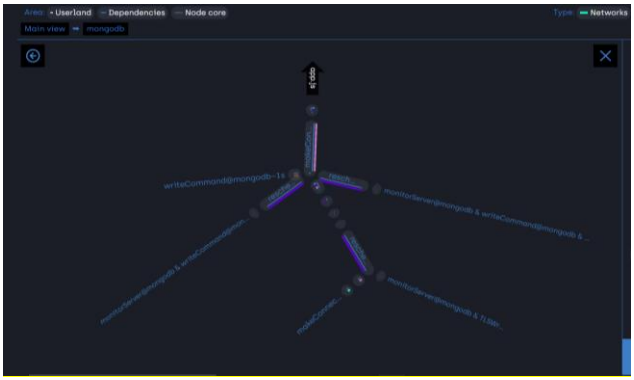

Figure 22 Profiling code bottlenecks with clinic.js database section is zoomed.

Figure 23 Profiling code bottlenecks with clinic.js HTTP connection section is zoomed.

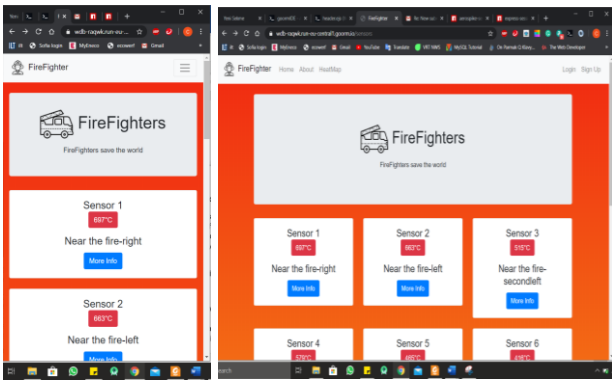

Figure 24 Front-end test of the webpage for smartphone and tablet

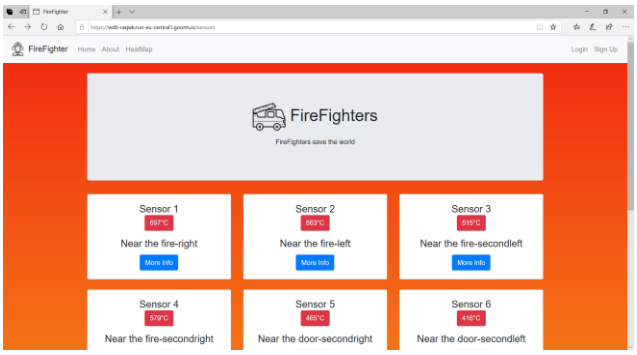

Figure 25 Testing with Edge explorer

Extra tests were performed with tablets and smartphones with different web engine. The website also responsive to them and it was not encountered any errors while using them.

[21] Matteo Collina,Finding Bottlenecks Retrieved 21 May from https://www.infoq.com/presentations/profile-nodejs/

5 Conclusion

The test results are up and coming. The results met customer requirements.

The website design is responsive to computers, smartphones, and tablets using.

The customer can monitor the sensor degrees in every 30s

The page load speed also met the requirements. Since the user will use a tablet during the training session, page load time will decrease.

Each sensor can be visualized deeply with a graph and heatmap.

The heatmap of the shipping container can be seen using the heatmap page. However, this section needs to be improved by working together with firefighters.

The restrictions were made by implementing the login and admin sections.

As it was mentioned in the previous section the MongoDB is a fast and efficient database. Moreover, it meets the expectancies for this project and the great advantage of MongoDB is that it does not require any database administrator. However, the project database can be switched to a relational database such as MySQL. The back-end design of the website provides an easy and practice transition to other types of databases.

The backend part of the website can carry the expected load. But extra improvement is needed. In this project, express-session was used in-memory store. This allocates and maintains a lot of memory (by design). It can be switched to a distributed session or JWT (JSON Web Token). Since the distributed session brings down the allocated memory, the website will be more responsive and faster.

6 References

Alex Coleman, High Level Design of a Web Application, Retrieved 15 May 2020 from https://selftaughtcoders.com/from-idea-to-launch/lesson-12/high-level-design-web-application-controllers-views/

https://developer.mozilla.org, Client-Server Overview, Retrieved 15 May 2020 from https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overviewMihir Shah, Retrieved 15 May from https://www.simform.com/mongodb-vs-mysql-databases/

cloud.mongodb.com,Collections Retrieved 15 May 2020 from https://cloud.mongodb.com/v2/5ea35f8f0b75ca4ba24f3442#metrics/replicaSet/5ea3601fabd5e7733b4d61dc/explorer

cloud.mongodb.com,Collections Retrieved 15 May 2020 from https://cloud.mongodb.com/v2/5ea35f8f0b75ca4ba24f3442#metrics/replicaSet/5ea3601fabd5e7733b4d61dc/explorer

docs.mongodb, Document Structure, Retrieved 17 May 2020 from https://docs.mongodb.com/manual/core/data-modeling-introduction/

docs.mongodb, References, Retrieved 17 May 2020 from https://docs.mongodb.com/manual/core/data-modeling-introduction/

https://docs.mongodb.com, Indexes, Retrieved 17 May 2020 from https://docs.mongodb.com/manual/indexes/

https://mongoosejs.com/, limit vs. perDocumentLimit Retrieved 17 May 2020 from https://mongoosejs.com/docs/populate.html

https://ejs.co/#docs, Tags, Retrieved 18 May 2020 from https://ejs.co/#docs

https://ejs.co/#docs, Includes, Retrieved 18 May 2020 from https://ejs.co/#docs

Patrick Wied, h337, Retrieved 19 May 2020 from https://www.patrick-wied.at/static/heatmapjs/docs.html

https://gtmetrix.com/, Analyze,Retrieved 19 May 2020 from https://gtmetrix.com/

https://gtmetrix.com/, Analyze,Retrieved 19 May 2020 from https://gtmetrix.com/

https://gtmetrix.com/, Page Speed ,Retrieved 19 May 2020 from https://gtmetrix.com/reports/wdb-raqwk.run-eu-central1.goorm.io/4izjqZ08

https://gtmetrix.com/, Page Speed ,Retrieved 19 May 2020 from https://gtmetrix.com/reports/wdb-raqwk.run-eu-central1.goorm.io/owe8TINt

https://gtmetrix.com/, Page Speed ,Retrieved 19 May 2020 from https://gtmetrix.com/reports/wdb-raqwk.run-eu-central1.goorm.io/4izjqZ08

Matteo Collina,A New Way To Profile Node.js, Retrieved 21 May from https://www.infoq.com/presentations/profile-nodejs/

Matteo Collina,Finding Bottlenecks Retrieved 21 May from https://www.infoq.com/presentations/profile-nodejs/

Matteo Collina,Finding Bottlenecks Retrieved 21 May from https://www.infoq.com/presentations/profile-nodejs/

7 Appendix

https://gitlab.groept.be/cemal.yetismis/firefighterwebsite GitLab link

https://kuleuven.mediaspace.kaltura.com/media/Firefigter4us/1_j79u8pt4 video of the website

https://floating-escarpment-11323.herokuapp.com/ with this link the website can be visited. The website was deployed to Heroku.

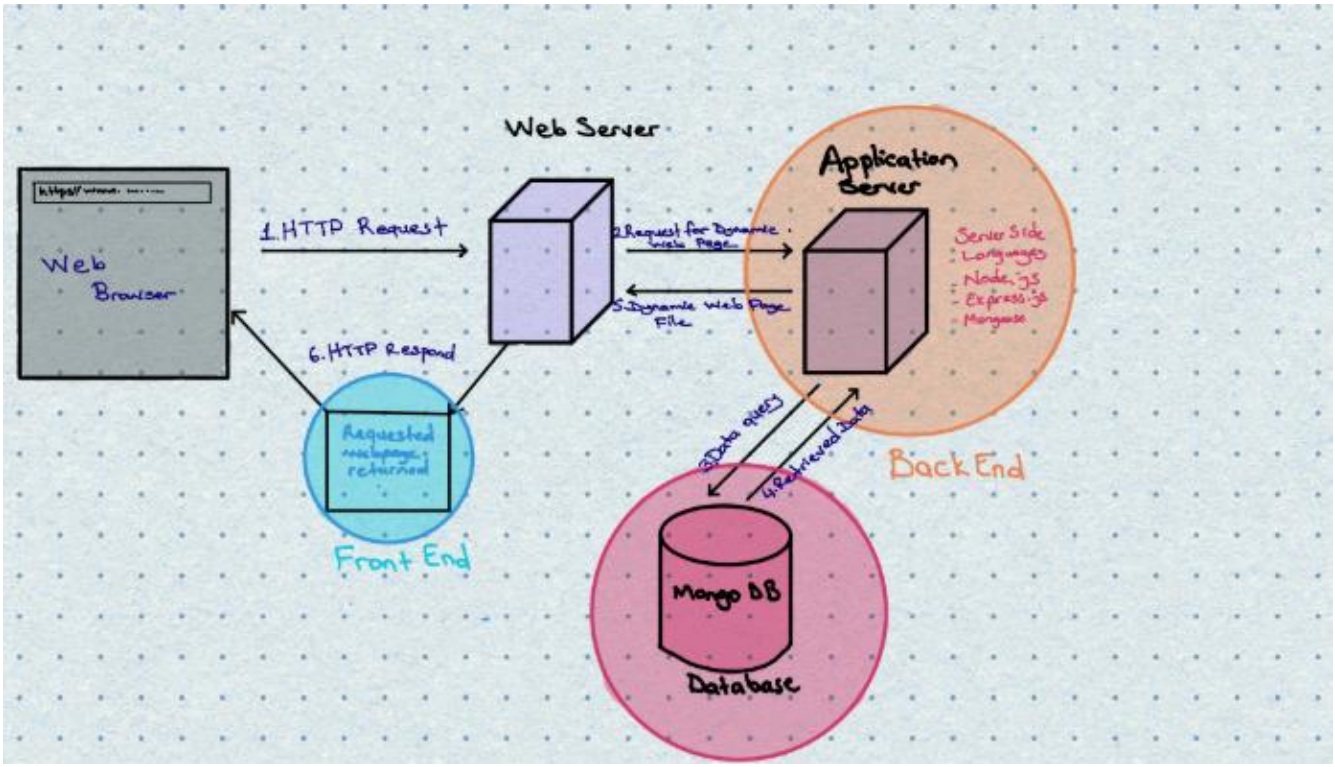

Figure 2: The Web side process to monitor the sensor's data

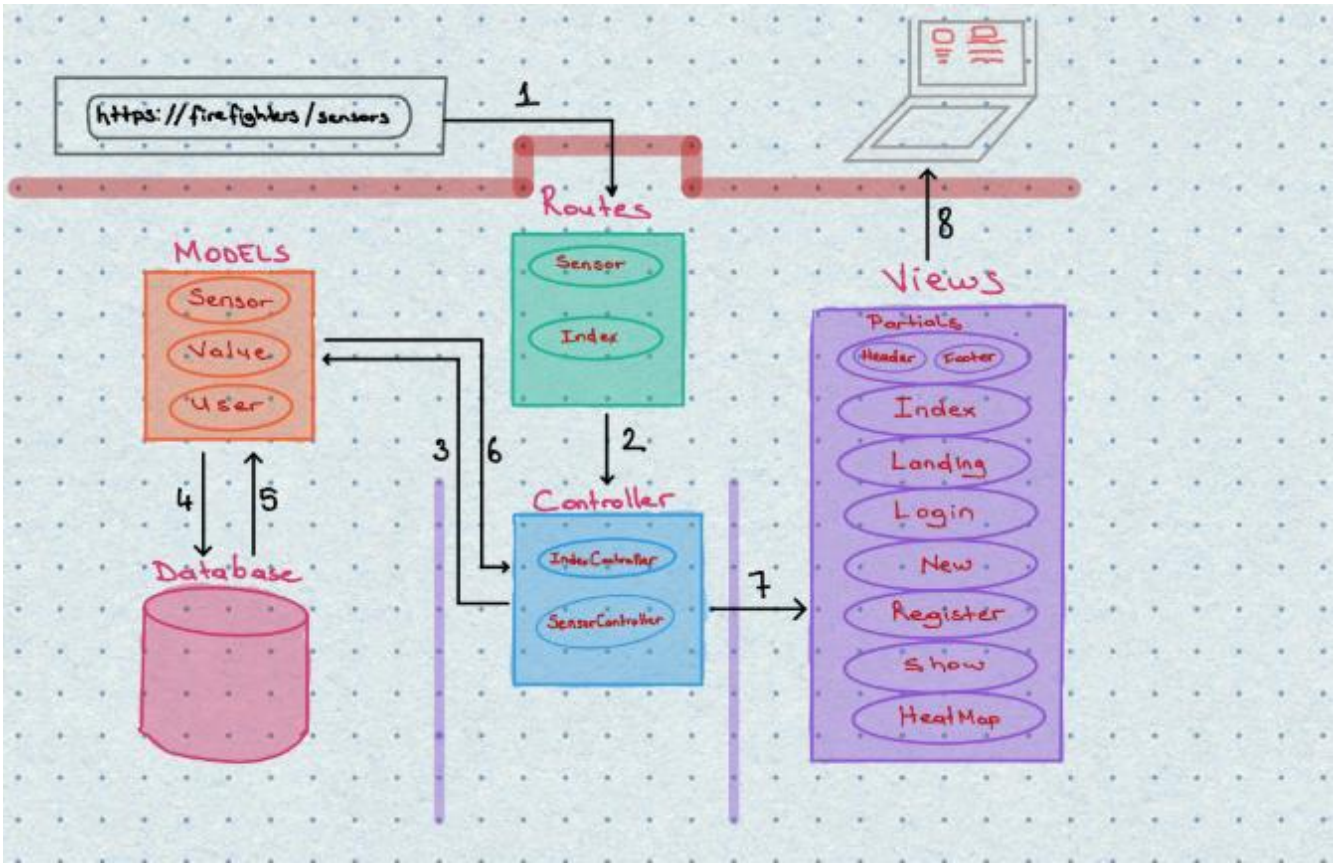

Figure 4: MVC implementation more detailed

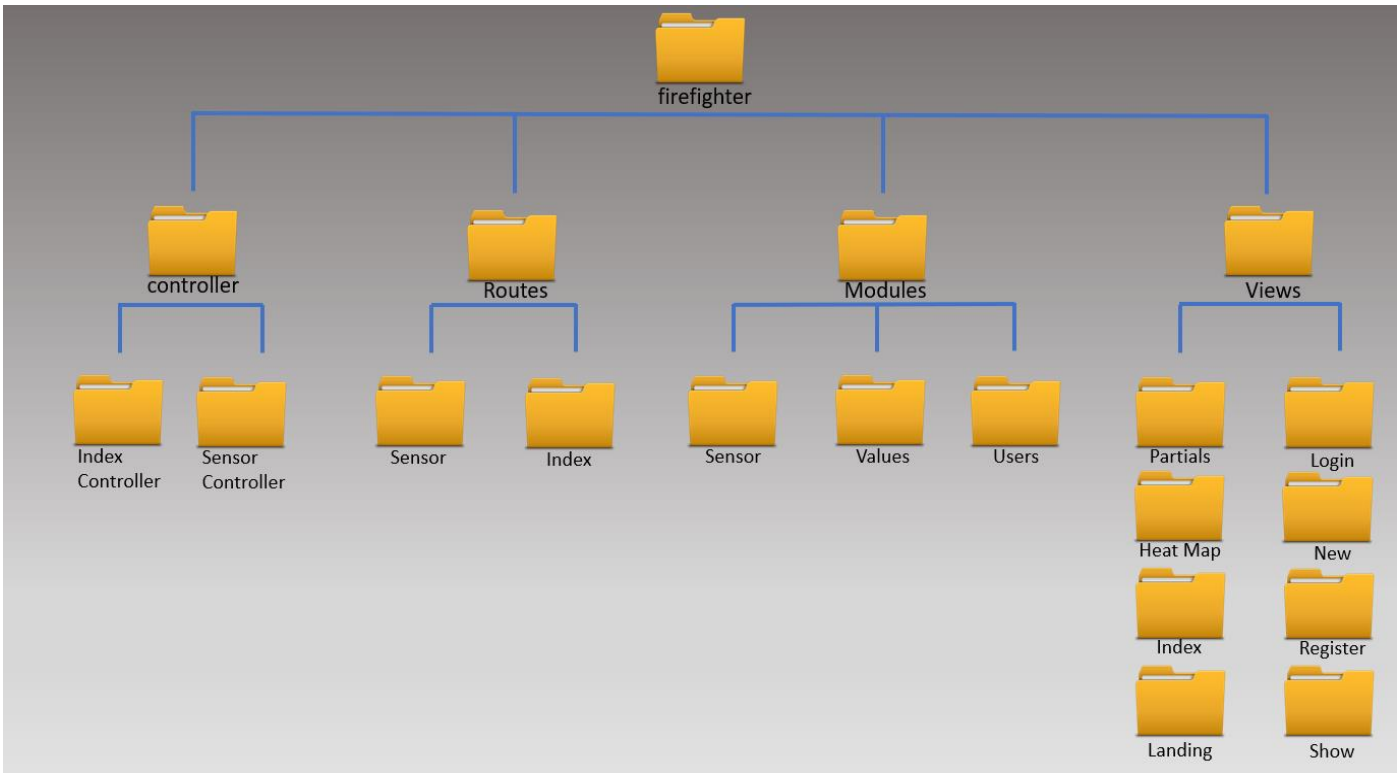

Figure 5: MVC implementation the web-site in as file tree

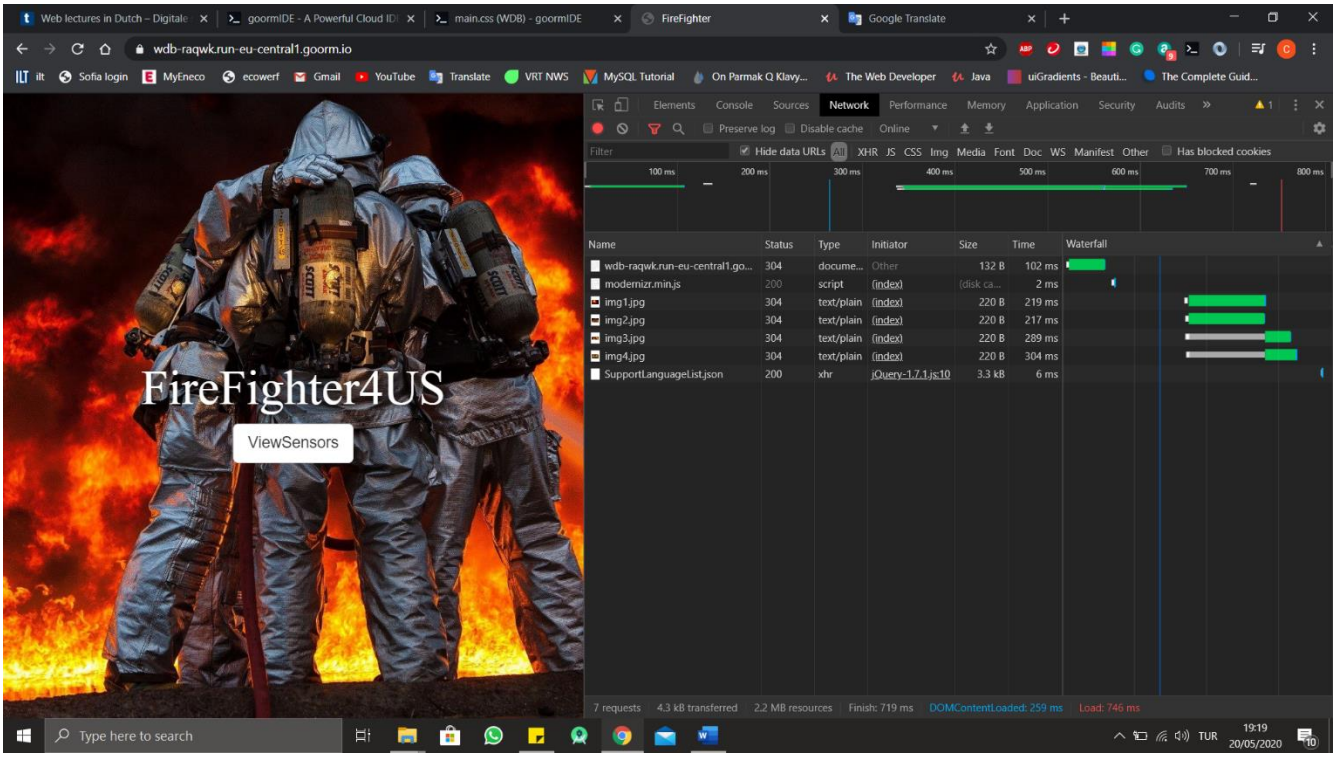

Figure 15 Performance and bandwidth measurement landing page

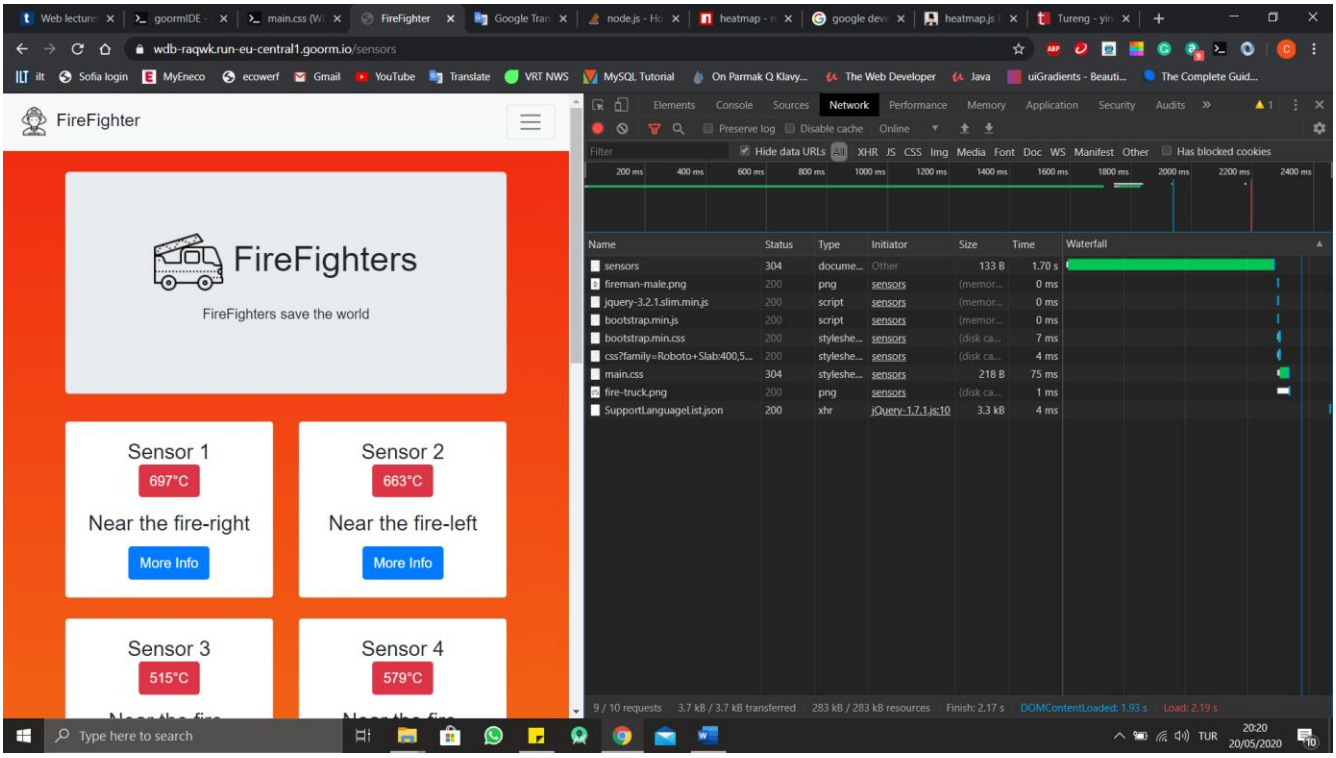

Figure-16 Network analyzing with google chrome tools Sensor page

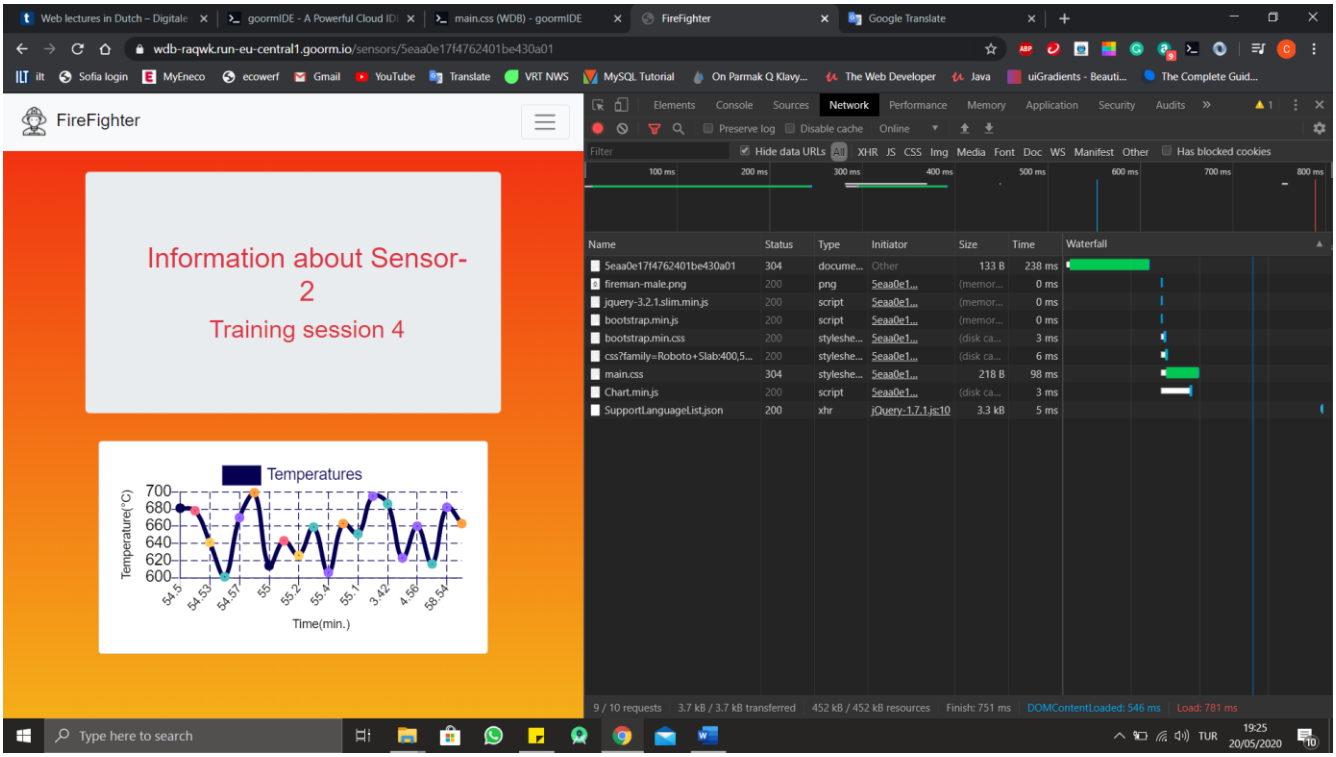

Figure-17 Network analyzing with google chrome tools Show page

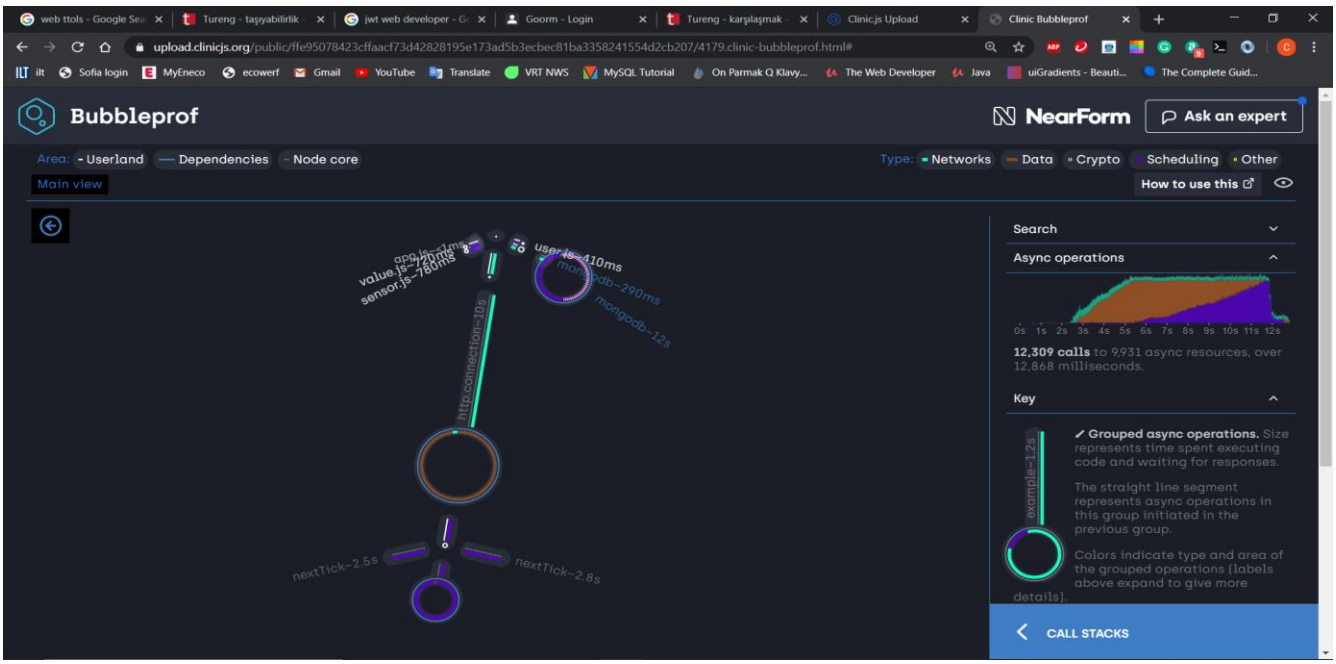

Figure 20 Profiling code bottlenecks with clinic.js

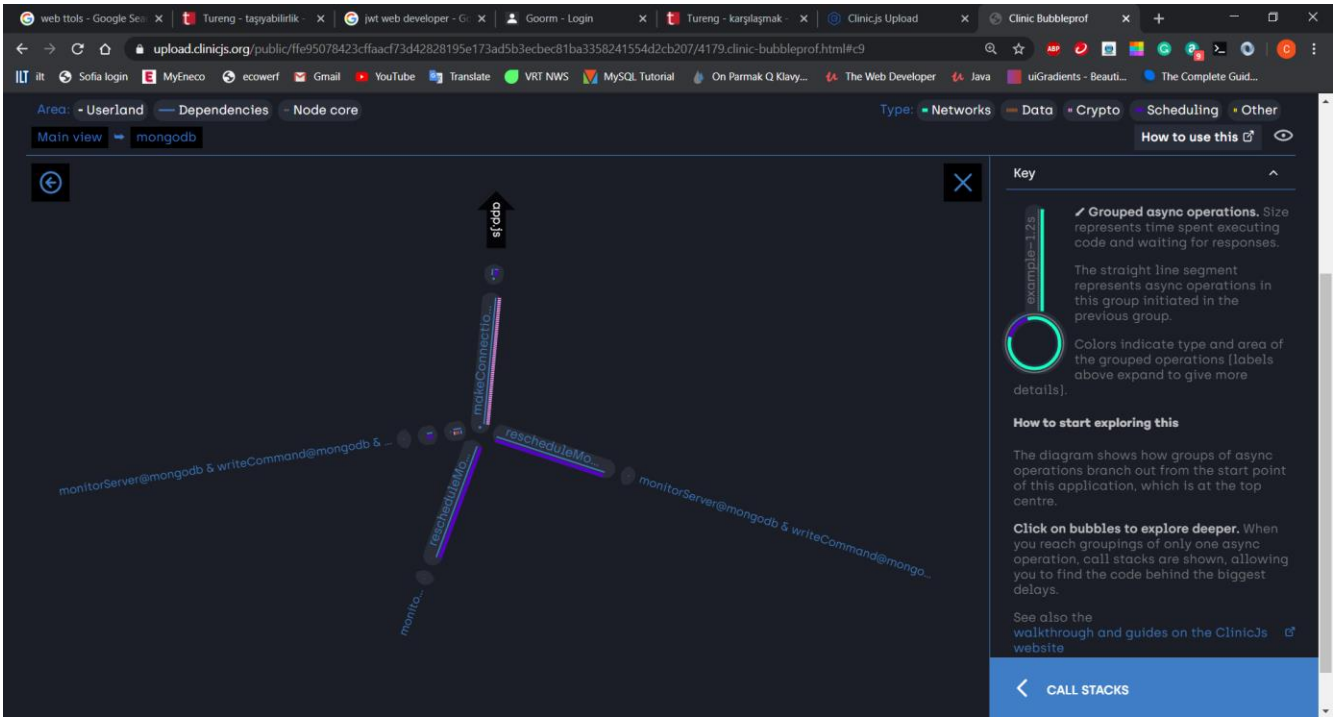

Figure 21 Profiling code bottlenecks with clinic.js database section is zoomed.

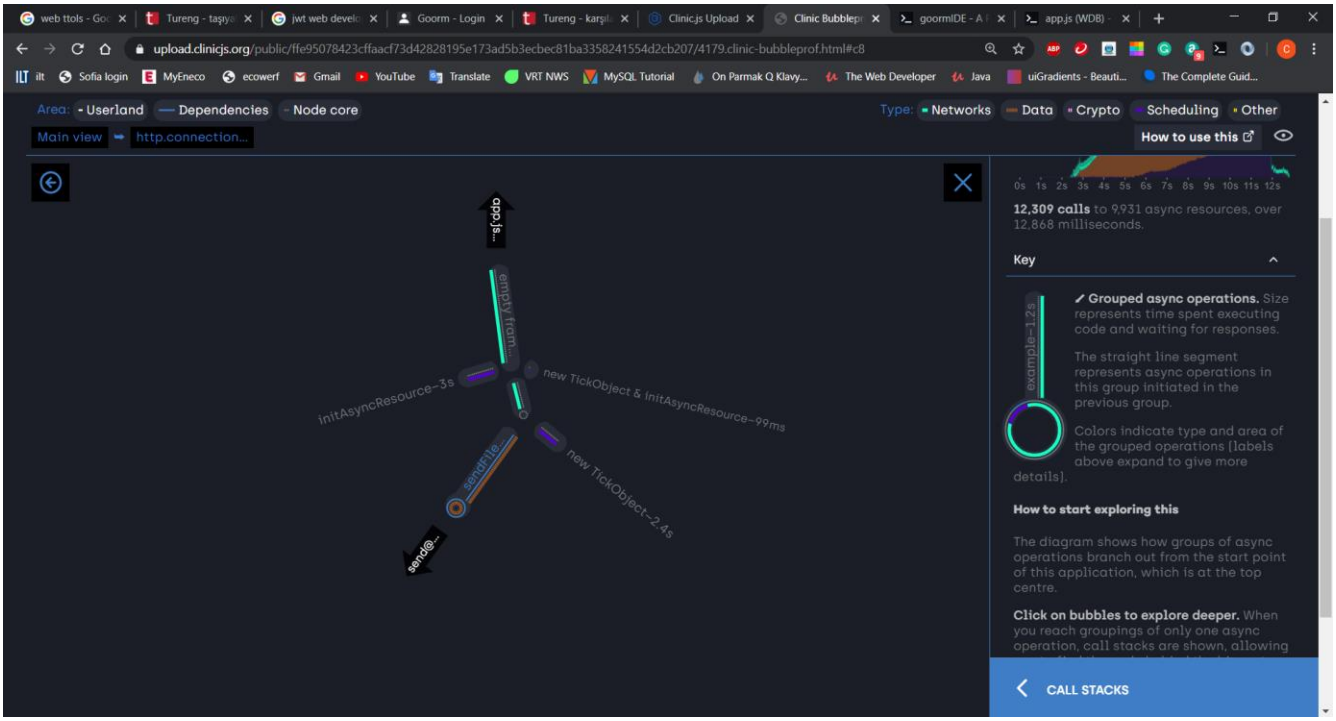

Figure 22 Profiling code bottlenecks with clinic.js HTTP connection section is zoomed.

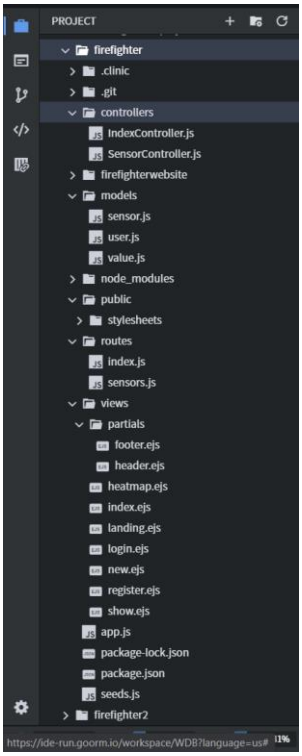

Figure-26 Folder view of the project

Figure-27 QR code of sensor page