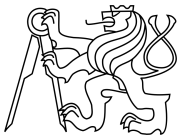
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF BACHELOR'S THESIS

Title:	Design and development of modules for real-time display of data from radiation monitors
Student:	Josef Vítovec
Supervisor:	doc. Ing. Carlos Humberto Granja, Ph.D.
Study Programme:	Informatics
Study Branch:	Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2016/17

Instructions

The aim of the thesis is to design and implement modules for data retrieval and data displaying from two different types of radiation monitors in the lab.

Functional requirements:
- Data will be displayed in real time.
- A user will be able to set the interval to display.
- A user will be able to set the interval and download appropriate data in both graphics and data formats.

Non-functional requirements:
- Modules will be designed as web components in order to be included into www presentation of the lab.

Procedure:
1. Specify and analyze requirements for data retrieval and display from both types of radiation monitors (devices).
2. Design the appropriate solution architecture including (i) communication with the devices and (ii) the local data storage structure.
3. Choose an appropriate implementation platform and implement the component for each type of radiation monitor and database.
4. Write documentation and test modules.

References

Will be provided by the supervisor.

L.S.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague February 2, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of software engineering

Bachelor's thesis

Design and development of modules for real-time display of data from radiation monitors

Josef Vítovec

Supervisor: doc. Ing. Carlos Humberto Granja, Ph.D.

17th May 2016

Acknowledgements

First and foremost, I wish to express my sincere thanks to my supervisor, doc. Ing. Carlos Humberto Granja, Ph.D., for offering me the ability to investigate such an intriguing topic.

I am also grateful to Vu Thien Trang, who not only helped me with the language corrective but has also been encouraging me during the writing of this bachelor thesis and supported me in the course of my university studies.

Finally, but first in my heart, my parents are due my deep gratitude for their continued moral and financial support throughout my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on 17th May 2016 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Vtovec, Josef. *Design and development of modules for real-time display of data from radiation monitors.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Tato bakalářská práce se zabývá analýzou, návrhem a implementací webové komponenty sloužící k vizualizaci dat v reálném čase. Webová komponenta umožňuje zobrazení dat ve formě dvou odlišných grafů, z nichž jeden zobrazuje data v reálném čase a druhý zobrazuje data v závislosti na zadaném časovém intervalu. Data mohou být pomocí této komponenty stahována, a to jak v textovém, tak v grafickém formátu. Zdrojem těchto dat jsou vědecká zařízení označována jako neutronové radiační monitory.

Klíčová slova vizualizace dat, real-time, MEAN stack, NoSQL, D3.js

Abstract

The primary purpose of this bachelor thesis is to deal with the analysis, design and the implementation of a web component, which visualizes the real-time data. The web component enables to display data in a form of two various graphs, the first graph displaying data in real-time, the second displaying data based on a given time interval. Thanks to the component, specific data might be retrieved, in both text and graphic format. The source of the data are the scientific instruments known as neutron radiation monitors.

Keywords data visualization, real-time, MEAN stack, NoSQL, D3.js

Contents

List of Figures

List of Tables

Introduction

The aim of this bachelor thesis is to design and implement the remote display of continious data from scientific instruments (neutron radiation monitors) as a real-time web component. The instrumentation and deployment of the developed application are part of the Van-de-Graaff particle accelerator laboratory of Institute of Experimental and Applied Physics of the Czech Technical University in Prague [5].

The main purpose of the component is to provide the remote real time display of data from the devices including customized settings in terms of data set and display range. The source of the data as starting point of this work are the read out electronics of the installed radiation monitors placed at the neutron targets in the accelerator. The resulting web component is intended to be included within the website of the accelerator laboratory which is planned to be developed.

Motivation

Sensors and electronic devices are widely used in industry and scientific laboratories including deployment in hazardous environments where presence of staff and operators is not possible. This is the case in radiation environments such as nuclear reactors and particle accelerators for electronic equipment installed which requires control and display in real time. The solution is to transfer the signals and data from the devices away for remote display. Concerning this work the instrumentation and necessary electronics are existing. What was needed was to extend the readout signals and data beyond the laboratory of the accelerator to make the information from the sensors (neutron radiation monitors) accesible to users and visitors of the laboratory.

Chapter 1

Analysis

1.1 Communication with radiation monitors

This section describes the data readout communication with the radiation monitors installed at the Van de Graaff accelerator. The communication consists of the transfer of readout data from radiation monitor to PC where the data are primarily stored. This step exists and works. The second step is to take the retrieved data beyond the laboratory to make it accessible elsewhere including off-site remote display via internet. A detailed description of this functional solution follows.

1.1.1 Data retrieval

The data transfer from a radiation monitor to local personal computer (which is located in a laboratory) is processed via a customized readout electronics device. This device controls, powers and reads out radiation monitor, from which it retrieves data and streams them afterwards through USB to the local PC. The data are then further relayed continuously from the local PC by C-based daemon[1] to external databases, where they are subsequently curated. An illustarting of this data chain is shown in Fig. 1.1.

1.1.2 Data specification

The data from the monitors are going to be sent to the database in packets of interval of a few seconds (for now we will take into consideration 5 seconds). The data format will be represented by two non-negative integers, alternatively by a vector (an array) of length 2 representing these two integers.

[1] a computer program that runs as a background process

1.1.3 Simulation

In order to properly test and develop the web component, it is valuable to simulate the data readout of the radiation monitor itself in a simulated expected environment. The task of simulating procedure will be done in C programming language in order to mimic and send data similar to those, which are supposed to be retrieved from radiation monitor, into an existing database. The mentioned C program will be later used for creating a daemon running on a local personal computer in the accelerator laboratory, in terms of real data transfer.

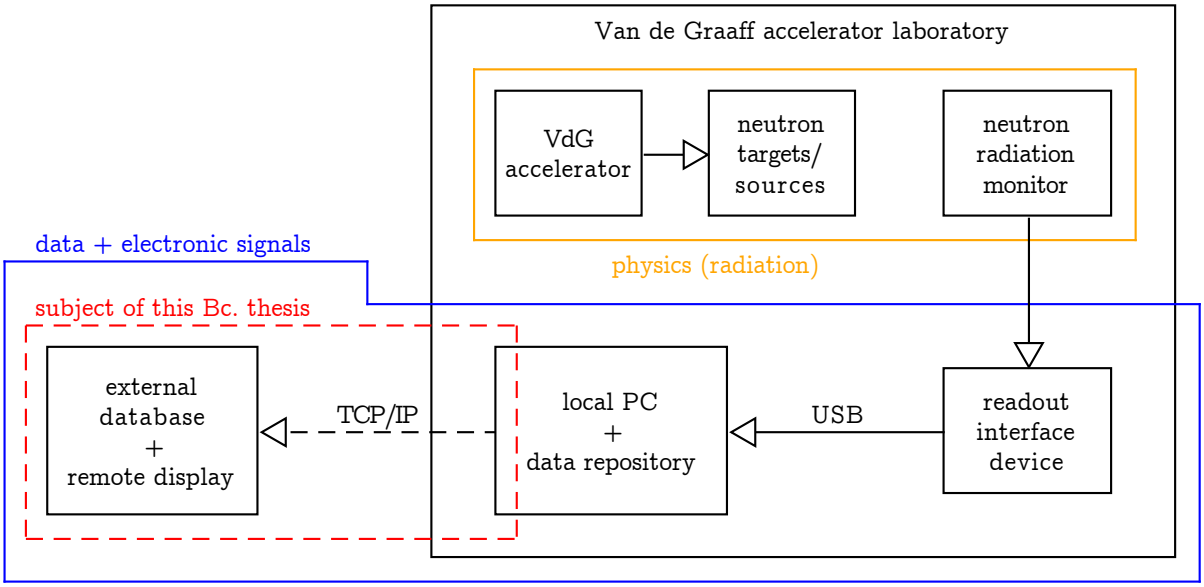


Figure 1.1: Diagram demonstrating data retrieval from radiation monitor

1.2 Requirements specification

Requirements specification validates the desired parametres regarding the component's functions and all the tasks it performs without taking into consideration an implementation. Requirements are divided into two basic categories. The first category consists of functional requirements, which describe the communication of the component with its periphery. The second category includes non-functional requirements, which specify the implementation methods [6]. Thus this sections lists and describes all functional and non-functional requirements.

1.2.1 Functional requirements

F.1 Real-time chart

The web component must contain a display in the form of a real-time chart, which continuously updates right after the arrival of the new data. The refresh rate is estimated at the 1-10 seconds rate.

F.2 Customizable chart

Furthermore, the desired web component ought to consist of a chart, which will display data in ranges (time period/interval) specified by the user. Hence the component should include a proper UI element (in the form of a date picker or a slider) which would enable the user to adjust and specify the data display by setting the time interval.

F.3 Downloading specified data

The user should be able to download the data displayed both in graphics file formats and text file formats according to the adjustable range settings. The graphics file formats requested are raster formats such as PNG or JPEG.

1.2.2 Non-Functional requirements

N.1 Database

Data needed for the web application must be stored in a persistent data storage. Since the data from the radiation monitors will be sent continuously, the size of the database should be limited. In consequence, the database implementation must ensure that the oldest data are gradually overwritten with the newly incoming ones.

N.2 Stability

Web component should be stable, which means that neither unexpected falls nor non-standard behaviour should occur.

N.3 Responsivity

The web component is intended to be a part of a web application devoted to the VdG accelerator laboratory. Although the future web application is not primarily intended to be viewed on mobile phones or tablets, it would be definitely suitable to develop the component in a way it will be independent on any specific resolution.

N.4 Charts properties

Considering the fact that the data will be accessed and used by scientists, the charts themselves must possess specific features.

N.4.1 Step chart

The step chart ought to be the chart type used for the implementation of both real-time chart and customizable chart. The step chart shows individual data changes in time in contrast to the regular line chart type, which displays merely the gradual progress. It is valuable to display the size of the bin which is useful in scientific applications.

N.4.2 Axis

The axis x will represent the time line of a range automatically set by default value (e. g. 10 minutes) and also adjustable by user settings. The axis y is supposed to be a logarhitmic one. These features apply to both real-time and customizable chart.

N.4.3 Legend

Both real-time and customizable charts should have a legend attribute consisting of the description of the displayed data. The axis must be accompanied by legend. The x-axis must be accompanied by units based on time (seconds/minutes) while y-axis by units based on count.

N.4.4 Normalization

Since the data will be stored in the database for several days, it might happen that the user would like to display a huge amount of data. Taking into consideration that the data will be retrieved from the radiation monitors every 5 seconds, so in three days there will be approximately 100 000 values available to display. In addition to possible negative impact on stability and fluency of the web component or even the whole website, the resulting graph might be poorly readable as well. Consequently, such an amount of data ought to be scaled in the first place and displayed afterwards.

Chapter 2

Design

After the requirements have been analysed and clearly stated, this next section contemplates a following step of the software development: the designing of the whole component. As has been stated in [6, p. 14], *design specifications faithfully render physical and logical structures that implement the requirements.* Consequently, this chapter's goal is to discuss the technologies and the whole component's architecture I have chosen in order to execute the implementation.

2.1 Architecture

The architecure of the whole web component will be three-tier, consists of:

- database
- server
- client

An advantage of such solution lies in the separateness of the tiers, which enables to develop these tiers without dependency on each other, moreover to develop each of them on another platform. And each of these tiers can be replaced without affecting the other ones.

2.2 Database

As it is necessary to opt for a database type, the central idea of this section outlines the issue regarding the data storage. Deducing from the requirements, the amount of data which are supposed to be stored in the database is going to be huge, up to 10^5 records. However, the structure of a single data record is not complex. The incoming data has the form of two integer values accompanied with a timestamp. Therefore the solution might lie in either opting for a classic

relation database or an alternative in a form of so-called NoSQL database. The following paragraphs will analyse the advantages of both options so that the final decision on picking a database type could be made.

2.2.1 Relational

A database organized in terms of the relational model (a relational database) still remains the most conventional database model in these days. A relational database uses SQL (Structured Query Language) as the database computer language for the retrieval and management of data stored in it. Data with identical features are gathered into individual tables. As a result, the data are collected in an organized way. Each data element is represented by one horizontal row of a table. All horizontal rows share the same amount and type of colums as shown in Table 2.1. The SQL enables to create relatively complex queries including various types of table joins, searching in the tables as well as using aggregate functions.

ID	Name	Surname	Height	Weight
45456	Hudson	Mohawke	181	84
47135	Kendrick	Lamar	185	73

Table 2.1: Data structured in a table in a relation database

One of the main features of the relational database model lies in the fact that it is connected with a set of properties known as ACID (Atomicity, Consistency, Isolation, Durability). This guarantees the database proper transactional processing, a database reliability and consistency, which are the key aspects of systems operating with sensitive information. Another advantage is undoubtedly the utilization of SQL itself, since SQL is a standard query language applied to all relational database models. In case it is necessary to move the system from one relational database to another one, it will not be any fundamental change [7].

2.2.2 NoSQL

As the amount of data collected keeps rising generally, it has become compelling to store these data in means other than the tabular relations used in a conventional relational database model. Correspondingly, this need has triggered a frequent use of the so-called NoSQL databases, which could be defined as databases that are non-relational and do not use SQL as their query language. One of the most familiar NoSQL databases are Cassandra, Redis, MongoDB and CouchDB. Apart from the relational database model, where the data are organized into tables of columns and rows, the vast majority of NoSQL databases models operate on a principle of a key and its value.

Precisely meaning, every value inserted into the database has a its proper key. If the retrieval of the value is required, the operation will be processed due to the key assigned to that value. It is essential to mention the fact that a value does not necessarily mean a string or an integer, it could represent an array or an object. Individual pairs of key and its value are gathered in documents. These documents might consist of a various number of these kinds of pairs and create collections. An example of a collection is shown in Fig. 2.1.

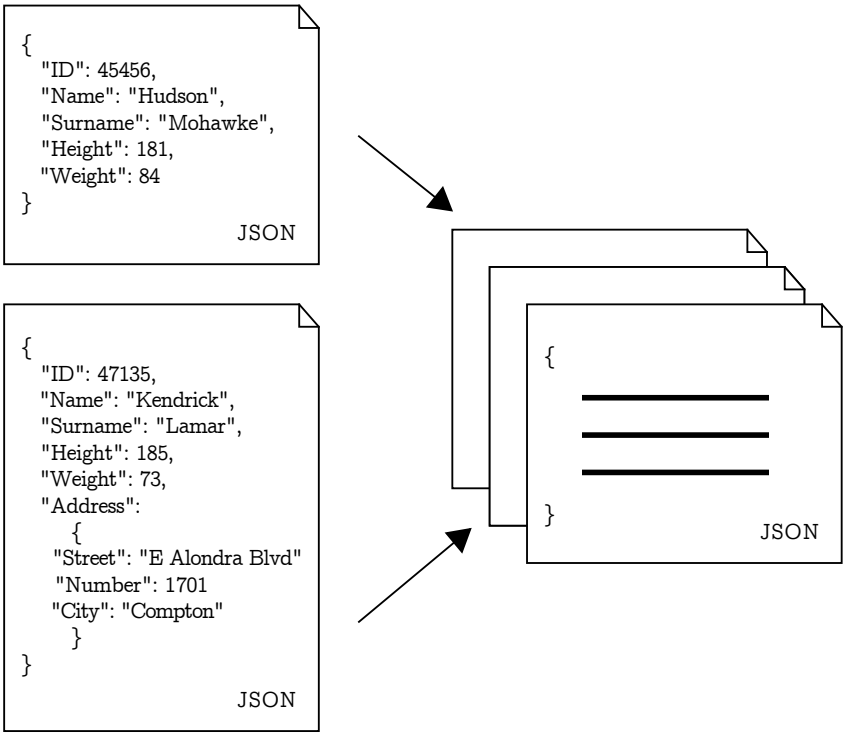


Figure 2.1: Data in a collection in MongoDB

The concept of NoSQL database model has a positive effect on a speed, some operations are simply faster using this approach, especially when it comes to a large amount of data which might not be completely structured/organized. The developers might find the concept of a key-value storage beneficial as well, moreover one could store whole object in this kind of database. The ACID model is replaced in NoSQL databases with the so-called BASE model prioritizing availability rather than the consistency of replicated data at write time[7].

2.2.3 Chosen solution

It is not straightforward to state which of the two potential solutions stated above is the better one. Each of them has its own use cases and it is dependent on each person to which extent they will identify these use cases applying to their particular system.

The database model type chosen for the purposes of this bachelor thesis is a NoSQL database MongoDB for several reasons. Firstly, hundreds of thousands of records could be considered as a relatively tremendous amount of data. Secondly, the final component is supposed to use insertions and quite trivial queries, MongoDB ought to be faster than a conventional relation database[8]. The use of the relational database would also be technically plausible in this case, but since the database will not hold any relations, it's meaningless. With the MongoDB usage, the option to profit from the dynamic schema could be taken into account. It will be possible to add some more data appropriate for display without altering the database structure. Last but not least, MongoDB is well-known for its feature called capped collection which noticeably simplifies the accomplishment of a requirement which is shown in section 1.2.2.

Capped collections are fixed-size collections that support high-throughput operations that insert and retrieve documents based on insertion order. Capped collections work in a way similar to circular buffers: once a collection fills its allocated space, it makes room for new documents by overwriting the oldest documents in the collection[9]. This behaviour is shown in Fig. 2.2.

2.3 Server

The following step of the design process is to opt for a server-side technology. I am not going to compare all the accessible server-side technologies. I shall weigh only two of them - specifically PHP and Node.js, since I have already encountered PHP in a form of Laravel framework and Node.js is a part of the so-called MEAN stack, which includes the MongoDB database (which has been already chosen as the database type) as well. Implementing the solution with Java has been taken out of question simply because of its robustness. I have no experience with Django and Ruby on Rails, which are frameworks of Python, or more precisely Ruby, thus the idea to use these technologies for the purpose of this bachelor thesis has also been eliminated. Both of the chosen technologies - PHP and Node.js will be compared in terms of requirements which have been stated in section 1.2.

2.3.1 PHP vs Node.js

PHP is currently the most frequently used language designed for web development and it runs on more than 80% of the world's web servers [10]. On the contrary, Node.js is a relatively new runtime environment that interprets

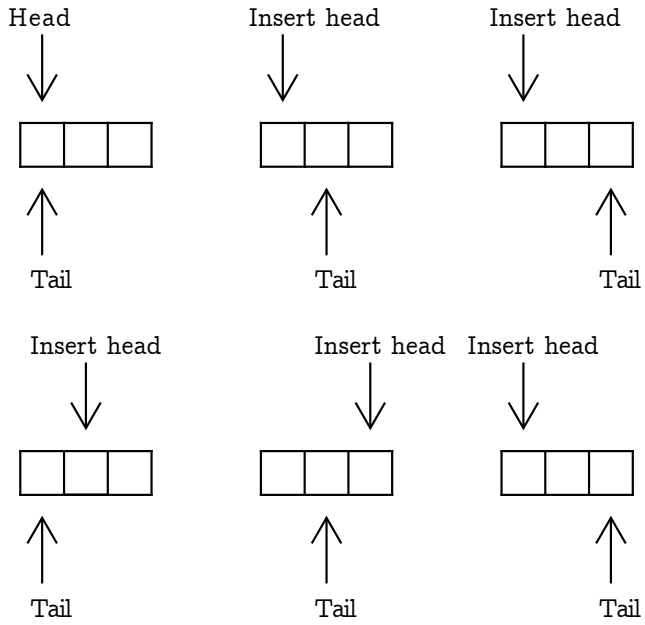


Figure 2.2: Behaviour of a capped collection. This digram shows collection of the length of 3 documents, where oldest document is replaced by newly incoming one when collection is full [1].

JavaScript using Google's V8 JavaScript engine. The consequence of PHP's widespread usage is that the vast majority of institutions maintaining websites also possess a server with Apache running on it and it is feasible to place a PHP application on such a server without any issues. This is the case of the IEAP, where a deployment of a Node.js application would require much more effort than a deployment of a PHP one. On the other hand, judging by the fact that Node.js is a more recent technology, it uses modern approaches reflecting the current trends in web development than PHP, which has been used for over 20 years in this field.

Node.js offers an easier creation of real-time applications thanks to the libraries such as Socket.IO[11]. Regarding the performance itself, Node.js ought to reach better results than PHP[12]. Another benefit pointing out that Node.js might be the solution is a simple manipulation with JSON format, which is also used in MongoDB.

Judging from the arguments stated above, I have decided to opt for Node.js as a server-side technology, as well as MEAN stack, despite the certain complicatons with deployment. So there will be a web application framework Express.js running on the server on the Node.js platform.

2.4 MEAN stack

MEAN stands for a software stack serving for the creation of dynamic websites and applications. MEAN stack consists of:

- MongoDB, a NoSQL database
- Express.js, a web application framework that runs on Node.js
- AngularJS, a Javascript MVC framework focused on development of SPA applications
- Node.js, a server-side runtime environment

It could be declared that the MEAN stack is a new competitor to a long-time familiar LAMP stack. Apart from the LAMP stack, MEAN stack comes with two major architectural changes. The first one lies in shifting from relation database model to NoSQL database whilst the second change observes a shift from server-side MVC model to the client-side SPA. A major advantage of MEAN stack is the usage of JavaScript accross the whole stack. The communication among MEAN stack components is shown in Fig. 2.3.

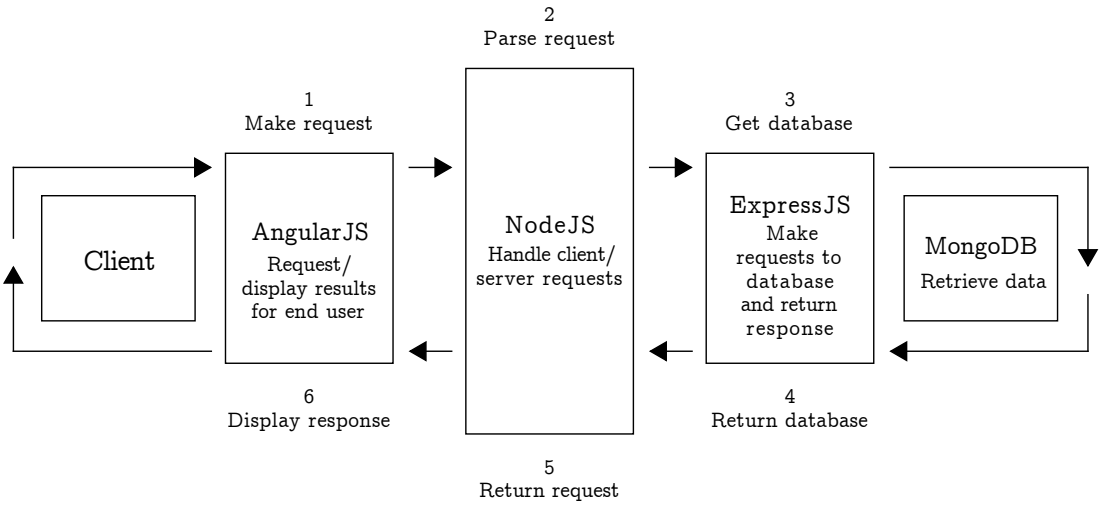


Figure 2.3: Communication between MEAN stack components [2]

2.5 Client

2.5.1 AngularJS

Along with the MEAN stack, a JavaScript front-end framework called AngularJS maintained by Google is used for creating the client. This framework is aimed primarily at SPA, which is also the case of the required web component. AngularJS brings several intriguing concepts such as two-way data binding, which keeps the model and view synchronized under all circumstances. Specifically, if a model is changed, the view gets updated at the same time and vice versa, as shown in Fig. 2.4.

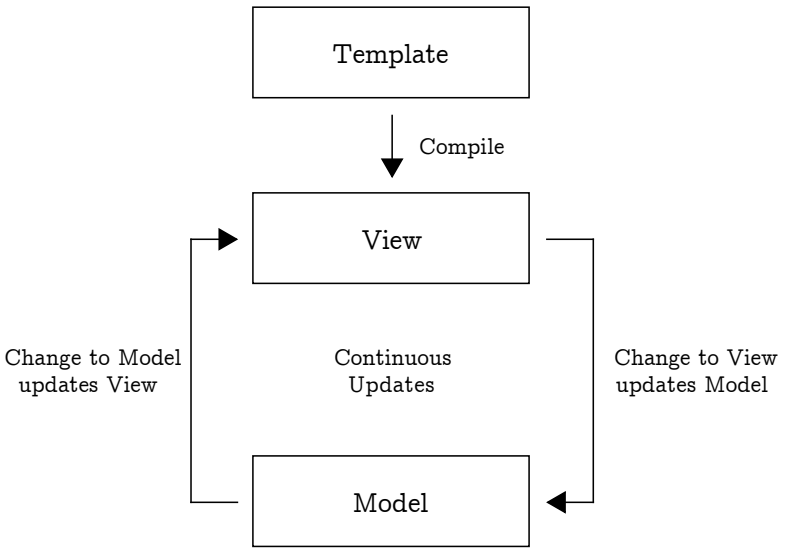


Figure 2.4: Two-way data binding in AngularJS [3]

2.5.2 Bootstrap

As I am experienced with working with Bootstrap, it will be used as a technology for prototyping of the application user interface. Bootstrap is HTML/CSS/JS framework which greatly simplifies development of responsive web applications. Moreover, this framework incorporates a lot of pre-crafted components that could be useful, such as various buttons, panes and navigational elements.

2.6 Visualization

Due to the fact that the primary feature of the required web components is to visualise certain data in a graph, it is necessary to pick a library which will enable me to do so.

2.6.1 D3.js

As I lack any experience with a library, which enables to visualise data in a graph, I have focused on these aspects: the extention and the requirements shown in section 1.2.2 regarding this library. Eventually, I have chosen the D3.js library. It is a JavaScript library, which title derives from merging the words Data-Driven Documents. D3.js is intended to create dynamic visualisations in web browser based on the data. As it has almost unlimited options regarding the data visualisation [2], it seems to be a good candidate.

However, the unlimited options of this library could be also considered as its huge disadvantages. If it was not implicitly compulsory to use a step chart, a chart with logarithmic axis y and a real-time chart with transitions, I would apparently use some of the solutions such as amCharts, Highcharts or Google Charts, which are provided by different libraries. They are easier to utilize, plus they contain a good size of pre-prepared graphs. Luckily thought, the community around D3.js is broad, hence it is possible to find a lot of examples describing the implementation of various visualisation.

Another drawback of the D3.js library to some other libraries that use HTML5 canvas for rendering is the fact that visualisations implemented in D3.js are not responsive by default, which is comprehensible according to the options offered by this library. Even though the responsive Boostrap framework will be utilised, the graphs themselves must be designed responsive individually, which should be viable[13].

2.7 Real-time

The goal of this section is to focus on the design of real-time communication demanded by the final web component in order to display the real-time chart. As I have already mentioned in section 1.2.2, the data are supposed to be sent in interval of 5 seconds. Needless to say, that the chosen solution should not be dependent on the knowledge of this interval. The interval should be changeable in the future without affecting the right function of real-time chart.

[2] `https://github.com/mbostock/d3/wiki/Gallery`

2.7.1 Polling

One of the options to take into consideration for realization of the real-time connection between the server and a client is polling[3]. In this case, polling would mean that a client would send a request once per x seconds and he would get an instant answer, as shown in Fig. 2.5. The issue of this solution lies in the constant interval itself, during which the request will be processed. If the length of the data sending interval was reduced, the data would be displayed with an immense delay. On the other side, if the interval was extended, a client would send some of the requests redundantly and the data would be displayed with delay as well.

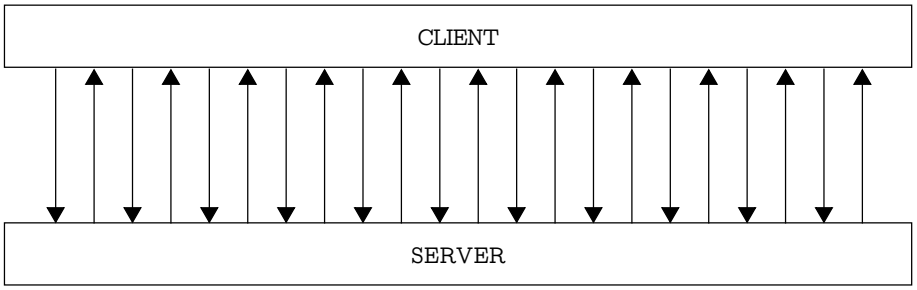


Figure 2.5: Communication between server and clients using polling[4]

2.7.2 Long Polling

Long-polling is an enhanced option of polling, which has been mentioned in the previous section and is also entitled as short-polling. Long-polling would mean that the client would send a request on a server and the server would leave the connection open unless having the answer for the client. After the client receives the answer, he immediately sends a request, this schema gets repeated, as shown in Fig. 2.6. This way not only the redundant request will get eliminated but also the delays emerging during the extending or the reducing of the interval of sending data from the radiation monitors. Even though this solution is close to real-time communication, a better solution than long-polling exists.

2.7.3 WebSockets

The best solution for the real-time communication in this case are undoubtedly WebSockets. WebSockets creates the TCP connection to server and keeps it as long as needed. This connection enables two-way data exchange in any time,

[3] a continuous checking the status of a device by client program

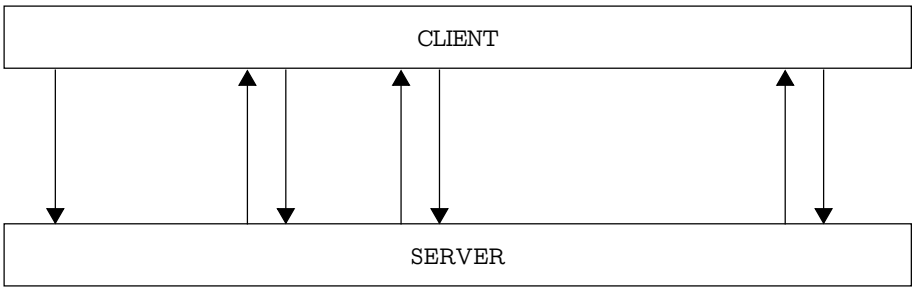


Figure 2.6: Communication between server and clients using long-polling [4]

as shown in Fig. 2.7. Both server or a client could cut off the connection. The main advantage compared to the polling methods is the unnecessity to create HTTP requests and responses. Headers of these request and responses then represent in a communication between server and a client a completely pointless network throughput, which is in comparison with the usage of WebSocket incredibly high. WebSockets also provide lower responses, which is caused by the fact that long-polling is forced to send a request right after obtaining a response. In WebSockets, the data can flow ceaselessly. Hence, a Node.js library called Socket.io will by used for the real-time communication between server and a client [14].

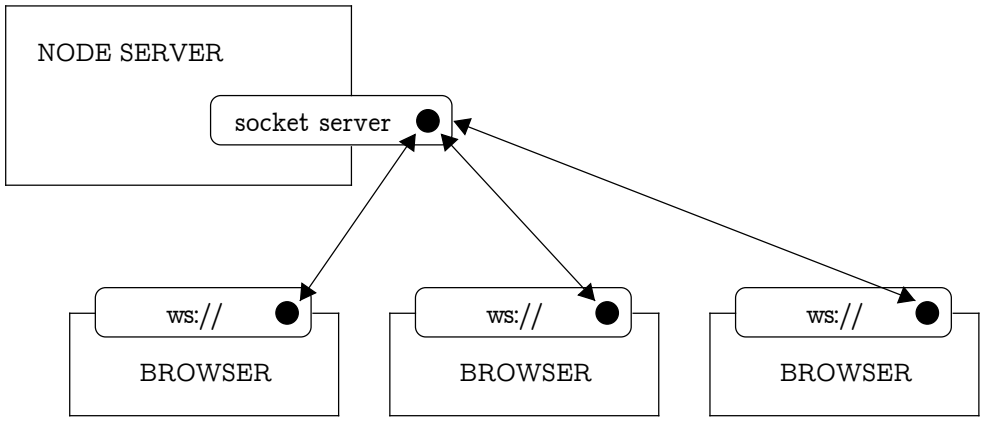


Figure 2.7: Communication between server and clients using WebSockets [4]

2.8 REST API

REST API serves for the data exchange between server and a client, which will be practiced primarily on customizable chart and the data download. REST is an interface architecture designed for distributed environment. REST is data-oriented, not procedure-oriented like SOAP or XML-RPC. REST defines an access to data via a set of four basic operations known as CRUD, that means Create, Retrieve, Update and Delete. Since the REST architecture utilizes HTTP, these methods fit to the corresponding methods of HTTP, specifically to the POST, GET, PUT and DELETE method[15].

2.9 Simulating the radiation monitor data readout

This section contemplates choosing the right solution regarding the simulation of the radiation monitor operation. The most appropriate solution for this issue seems to be the MongoDB C Driver[16]. MongoDB C Driver is an oficially supported and well-documented driver for MongoDB. C Driver offers a convenient way to connect to an already existing MongoDB database or to perform fundamental CRUD operations in the database. For the needs of this bachelor thesis, creating documents will be sufficient. Since the documents in MongoDB are stored in a BSON[4] format, it is necessary to create them in C in the same format. This will be achieved by employing the libbson library. The basic usage of this library is included in the MongoDB C Driver documentation.

2.10 User interface design

Even though the user interface of the web component will not be any complex, it would be suitable to create at least a basic prototype of the component. The prototyping tool chosen for these purposes is Axure RP. It could be defined as a complex tool providing a set of advanced functions such as the conditional display of the screen parts or clicking through to another screen. Creating the basic prototype will be efficient due to the large amount of reusable components, wide range of formatting options and the drag-and-drop function.

The whole component ought to consist of three main containers. The first container will be set for real-time chart while the second container will hold a customizable chart. The third container will contain two input fields. Clicking on these input fields, the date-time picker should appear via which the user could choose the interval. Based on the chosen intervals the user would be able to click on one of the buttons, which will trigger an action corresponding to it. Basic prototype of the component is shown in Fig. 2.8.

[4] a binary-encoded serialization of JSON-like documents

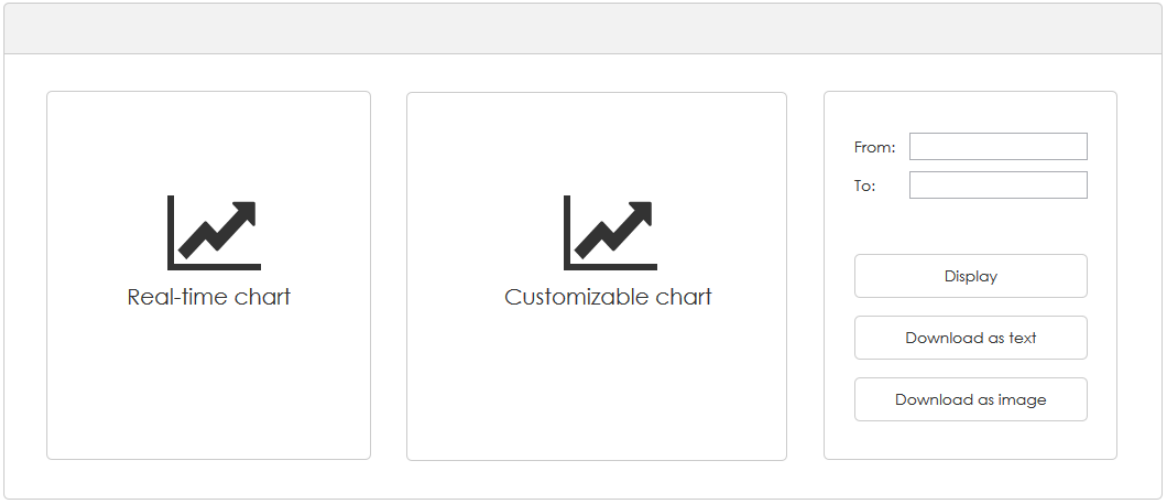


Figure 2.8: Basic prototype of the component.

Chapter 3

Realisation

This chapter describes an implementation of the web component. Instead of depicting the process in a detail as a whole, it would be appropriate to focus in the first place on the most significant parts of the application and to focus on parts, which inevitably need a further explanation.

3.1 Database

The one and only task to be done using the MongoDB shell was to set up a database. That has been accomplished with a command called `USE`. Other parts regarding the database have been implemented on a server, thus the following section is going to provide further information.

3.2 Server

The server is supposed to provide two functionalities to the application. The first functionality of the server will enable data transfer via REST API while the second functionality will ensure a communication with clients via Web-Sockets.

In order to operate with a database, a Node.js package Mongoose will be utilized. Mongoose is an object data mapping (ODM) library, which provides an integration of Node.js with MongoDB and simultaneously transforms data in the database to JavaScript objects for the use in an application. Its principles are similar to those of ORM (object relational mapping), which enables data conversion between relational database and a database runned with object oriented programming language.

Due to the possible work with data, which will be stored in a database, it is essential to create schemes and models. Schemes define the data structures in individual documents and models represent these documents, which could be later used for the data storage and data retrieval. In this case, every document

will represent a signal sent from a radiation monitor. As has been mentioned in the requirements, data sent from a radiation monitor will have a form of two integer values. The Mongoose also provides the enrich the schema with several optional parameters such as the collection name. Along with applying this feature, I have added another parameter to the collection which states that the collection will be capped and of a fixed length 65536 bytes. The code is shown in Listing 3.1.

```
1  var mongoose = require('mongoose');
2  var Schema = mongoose.SchemaM;
3
4  var SignalSchema = new Schema({
5    value1: {
6      type: Number,
7      required: true
8    },
9    value2: {
10     type: Number,
11     required: true
12   }
13 }, { collection: 'signal', capped: 65536 });
14
15 module.exports = mongoose.model('Schema', SignalSchema);
```

Listing 3.1: A mongoose model representing a signal from radiation monitor

3.2.1 REST API

Once the model is defined, the next stage that needs to be described is the creation of REST API. The REST API is closely connected with the concept of CORS. CORS (Cross-origin resource sharing) allows to reach resources of the website from other web domains. In the case of the web component, the REST API on the server could be requested from another domain, which will be represented by a client tier able to request it. CORS support in Express.js shown in Listing 3.2.

```
1  var app = express();
2
3  app.use(function(req, res, next) {
4    res.header('Access-Control-Allow-Origin', 'http://localhost
         :9000');
5    res.header('Access-Control-Allow-Methods', 'GET');
6    res.header('Access-Control-Allow-Headers', 'Content-Type');
7    next();
8  });
```

Listing 3.2: CORS support

In order to create REST API itself, I am going to use an Express.js library called node-restful. If a Mongoose model is passed to this library, it automatically generates RESTful routes. Even though the only method to be used for the purpose of web component is the library's GET method, it is of a such simplicity and a power that I have decided to use it, as shown in Listing 3.3.

```
1  var restful = require('node-restful');
2
3    restful: function(app, route) {
4
5      var rest = restful.model(
6        'Signal',
7        app.models.signal
8      ).methods(['get']);
9
10     rest.register(app, route);
11
12     return function(req, res, next) {
13       next();
14     };
15   }
```

Listing 3.3: Usage of node-restful library

3.2.2 WebSockets

As I have mentioned before in subsection 2.7.3, since the web component is desired to be real-time, the mechanism of WebSockets is presumed to handle this requirement. Precisely, Socket.io, the JavaScript Node.js library using the concept of WebSockets, has been selected. Apart from offering standard events such as connect or disconnect, Socket.io provides an option to emit and receive custom events. The library usage, especially the usage of its basic events - connect and disconnect - is shown in Listing 3.4. The method `streamSignals` is going to be explained below.

To sustain the real-time feature of the web component, it is crucial to capture the moment, when new data arrive to the database (a collection). This could be done by using the so-called tailable cursors. Tailable cursor is the MongoDB feature intended to be used for capped collections. A cursor present in MongoDB generally works in a way that it queries data from collection and it automatically closes, when there are no data present in the collection. On the contrary, the tailable cursor remains still open, ready for the arrival of the new data. If the new data arrives into the collection after some time, the tailable cursor is capable to retrieve them [9].

This functionality is included in Mongoose as well, which tails the collection based on a given generated model, in this case the Signal model. The

newly incoming data are sent afterwards to the client thanks to the Socket.io library. [17].

```
1  var app = express();
2  var server = http.Server(app);
3  var io = require('socket.io')(server);
4  var SignalCtrl = require('./controllers/SignalController');
5
6  io.on('connection', function(socket) {
7    console.log('user connected');
8
9    SignalCtrl.streamSignals(socket);
10
11   socket.on('disconnect', function(){
12       console.log('user disconnected');
13   });
14 });
```

Listing 3.4: Usage of socket.io

Tailing of the whole collection is strongly unrecommended though. In case the web component is being loaded for the first time, the tailable cursor needs to stream the whole collection prior to accessing the newest data, which are supposed to get displayed. Hence the tailable cursor has to go through all the records one by one, which might take up to tens of seconds. Once the web component has been loaded, the tailable cursor should proceed without any issues, however the first load would be time-consuming and the stability of the whole component would be exposed to risk. After taking this fact into consideration I have decided to query the newest record merely, on which the tailable cursor with stream is applied afterwards. The whole process including the data transmission to the client-side using the socket.io is show at Listing 3.5 in a method `streamSignals`.

All documents which are added to collections in MongoDB automatically consist of an `_id`, which is unique and which represents a document's primary key. Capped collections are no exception in this. The advantage of this `_id` lies in the fact that it contains a hidden timestamp, which can be retrieved by the method `getTimestamp()`. Hence, the timestamp is what is being sent to the client-side instead of `_id` as it is more suitable for later operations with it.

3.3 Client

This section focuses on the issue of data retrieval from the server, considering the data from REST API as well as the real-time data obtained via Socket.io. The process of the data visualisation itself would be depicted in this section as well.

For the creation of the client side, yeoman AngularJS generator [18] has been used, as it is intended for generating the scaffolding of new application. This generator produces the fundamental structure and the configuration of the application, which saves a great deal of effort and time.

```
1  var Signal = require('../models/signal');
2
3  streamSignals: function(socket) {
4
5      var date = new Date();
6      date.setSeconds(date.getSeconds() - 5);
7
8      var stream = Signal.find({ _id: { $gt: dateToObjectId(date) }
           }).tailable().stream();
9
10     stream.on('data', function(data) {
11       data = {
12         value1: data.value1,
13         value2: data.value2,
14         timestamp: data._id.getTimestamp()
15       };
16
17       socket.emit('newValue', data);
18     });
19 }
```

Listing 3.5: Merely the last record is chosen from the collection, a tailable cursor is applied to it and then streamed. A listener of the stream captures the incoming data which are then sent to the client-side.

In order to reach an already created REST API, I have opted for the Restangular library. Restangular is AngularJS service, which facilitates the creation of GET/PUT/POST/DELETE request. Hence it is an ideal solution for the data consumption from REST API. Restangular is simultaneously well-documented and effortless to use. Regarding the configuration of Restangular, it is indispensable to point to the REST API server. The factory called Signal is created afterwards, which will get the data from REST API. This is shown in Listing 3.6.

```
1  .config(function (RestangularProvider) {
2      RestangularProvider.setBaseUrl('http://localhost:3000');
3  })
4
5  .factory('Signal', function(SignalRestangular) {
6    return SignalRestangular.service('signal');
7  })
```

Listing 3.6: Restangular configuration

A similar principle is implemented on a client-side by using the Socket.io library, which serves for retrieving the data from server. The factory is also present in this case, and it contains a defined URL that represents the point of connection. Thanks to this factory, the data could be retrieved in a controller and then processed afterwards.

3.3.1 Visualization

The chart implementation has been done with the sources of a book D3 on AngularJS [19], which depicts best practices how to combine the D3 library with AngularJS. Both charts are implemented as custom directives[5], retrieving the data from the controller.

3.3.1.1 Real-time chart

To give a real-time chart a form, I have been inspired by an article [20], in which the author of the D3.js library demonstrates himself how to create real-time charts.

If the new data arrive to the database, the change is drawn to the server-side via Socket.io. Afterwards, the data are sent from the server-side to the client-side, where the data in controller are passed into a variable. This variable is then observed by a directive, which calls a function updating the chart every time there is an alteration, this is shown in Listing 3.7.

```
1 scope.$watch('val', function(newData, oldData) {
2     if(newData) {
3       updateData(newData);
4     }
5 })
```

Listing 3.7: Watching a variable with new data

The updating function itself then redraws the x asis and the two lines and leads to a transition, which delivers an impression of animation. The data intended for the visualisation are stored in an array of a fixed length. The newest arriving data values replace the oldest data values. The whole chart is shown in 3.1.

3.3.1.2 Customizable chart

The customizable chart features have been enriched with the option of focus + context zooming [21]. User might choose a time interval determining the data display via datetime picker. After clicking the button `Display`, the data are displayed. The whole chart is composed of two charts - one chart is greater

[5] markers on a DOM element which attach a special behavior to it

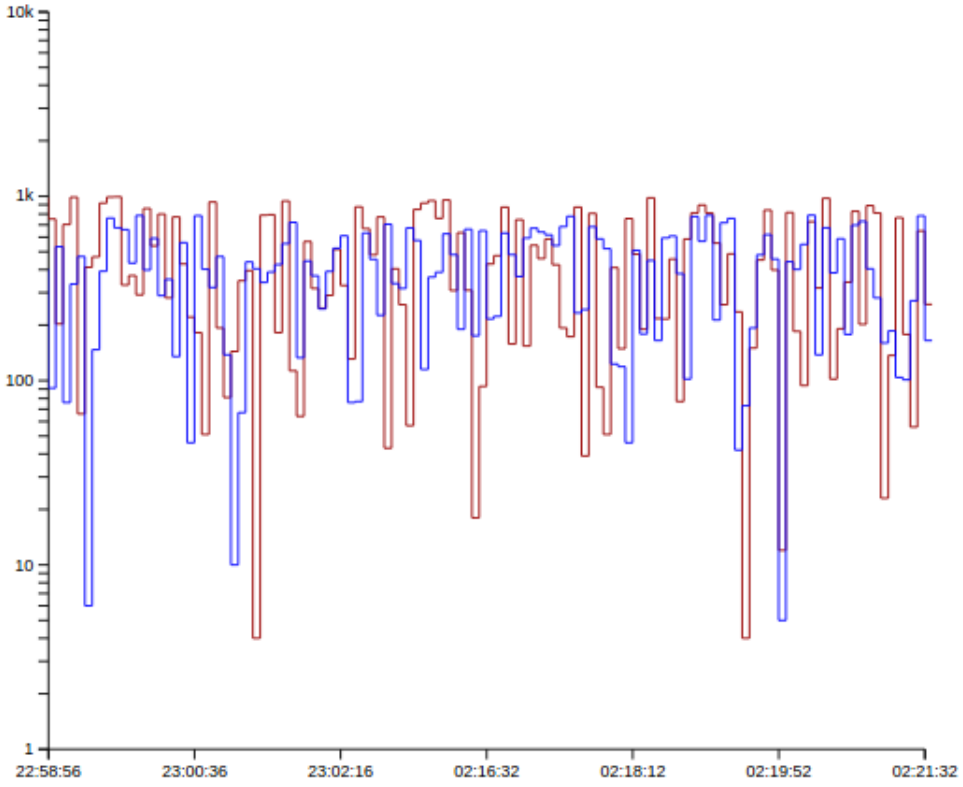


Figure 3.1: Real-time chart

than another. The smaller chart gives the user an ability to choose various time intervals, the data are then displayed in the second chart in detail while the smaller chart still holds the view of the chosen time interval. This chart is shown in 3.2.

3.4 Data retrieval

The web component ought to provide data retrieval in text file formats as well as the graphics file formats.

3.4.1 Text format

Data retrieval in the text file format is enabled by datetime picker, where the user chooses the time interval before downloading the data by clicking on a button. For the implementation of data retrieval, I have decided to use Angular File Saver, which leverages the well-known FileSaver.js library, which serves for file save on a client-side.

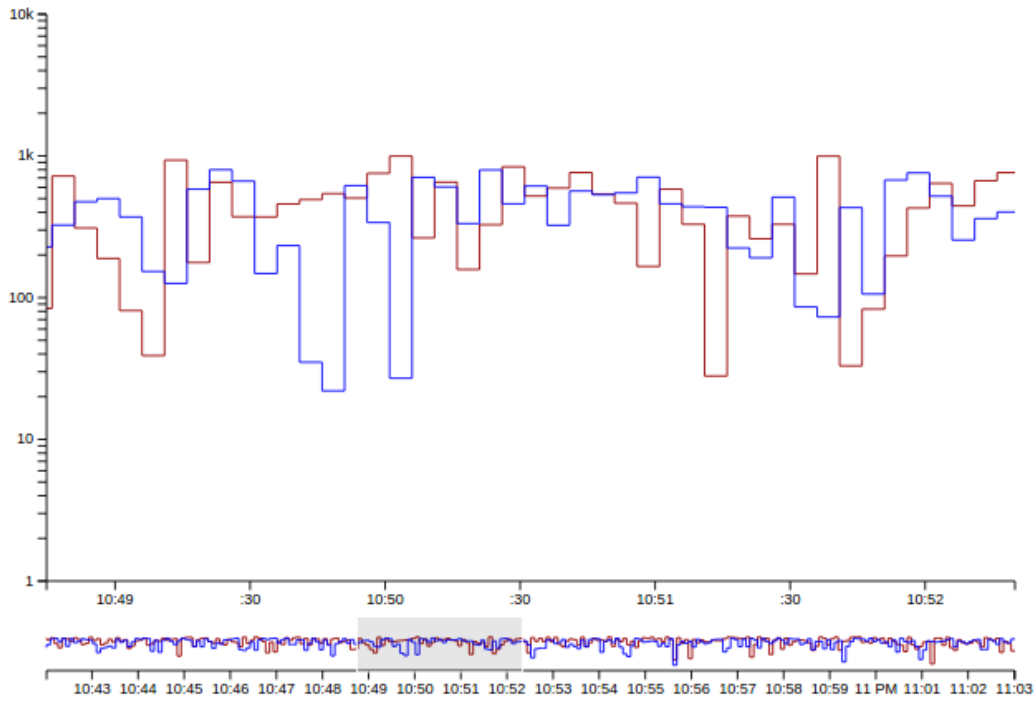


Figure 3.2: Customizable chart with focus

3.4.2 Graphics file format

D3.js charts are rendered as SVG elements - the elements of a vector image format. The requirements consist of raster graphics file formats, which means that an svg element containing chart needs to be converted.

In order to find a solution to this issue, I was inspired by the article [22]. Even though it refers to the Highcharts library, the concept is similar. Primarily, the code of the particular SVG element should be retrieved. This SVG element ought to be rendered to a canvas element, which has a textttoDataURL function. This function extracts the canvas content to an image and returns the data URI along with the image. This URI is then downloadable as a PNG image. However, this method deals with ordinary SVG elements, not with a created D3.js chart.

Owing to the fact that D3.js is a rather specific library, I was trying to find a solution especially for this library. Eventually, I have found an example of [23], which deals with a similar situation. It exploits from a bookmarklet[6] SVG Crowbar, whose role is to extract SVG elements from a web site and to enable its users to download them as SVG files.

[6] a small software application stored as a bookmark in a web browser

The example portrayed above benefited from the features of the bookmarklet SVG Crowbar, ze kterch vak vytv vlastn een, kter umouje retrieve SVG elements as PNG images.

The example portrayed above benefited from the features of the bookmarklet including the possibility to retrieve SVG elements as PNG images. The drawback of this solution is the fact that the retrieval is possible merely in the Google Chrome browser, not in Mozilla Firefox (other browsers have not been tested). Clicking on a retrieval button in Mozilla Firefox does not trigger any action. There is not any error message present in the console log, which disables any potential dealing with this issue. The retrieval of a PNG image (in Google Chrome) lies on a principle that after clicking on button `Download as PNG`, the current content of the customizable chart is retrieved.

3.5 Simulation

This section focuses on realisation of data simulation. As has been mentioned before, the data are supposed to be coming from the radiation monitors. The simulation of this phenomenon has been applied with the MongoDB C Driver, resulting in a C program which is attached to a capped collection in the database. The database receives randomly generated data in a form of two integers every 5 seconds. Generating data is shown in Listing 3.8.

```
1  while(true) {
2    randomValue1 = rand() % 1000 + 1;
3    randomValue2 = rand() % 800 + 1;
4    doc = bson_new();
5    bson_oid_init(&oid, NULL);
6    BSON_APPEND_OID(doc, "_id", &oid);
7    BSON_APPEND_INT32(doc, "value1", randomValue1);
8    BSON_APPEND_INT32(doc, "value2", randomValue2);
9
10   if(!mongoc_collection_insert(collection, MONGOC_INSERT_NONE,
         doc, NULL, &error)) {
11     fprintf(stderr, "%s\n", error.message);
12   }
13
14   bson_destroy(doc);
15   sleep(5);
16 }
```

Listing 3.8: Loop in which data are being generated and transmitted to database

Chapter 4

Testing

Testing is a crucial step in the software development. The earlier an error is identified in the development process, the less expensive it takes to repair it. Along with the development of the web component, the following tests have been proceeded: the unit tests and the functional tests. The reason behind choosing these two testing types is that the designed web component does not comprise any great deal of logic and communicates merely with the server, from where it retrieves data.

4.1 Unit tests

Unit tests are a type of automated tests testing individual units of short code. This kind of tests are usually created by programmers in order to test whether the new or the altered part of the code works as expected.

In terms of the web component, the unit tests were proceeded as continous tests of smaller sets of code on a client-side. The vast majority of the aspects the unit tests checked were the limit values which might potentially cause the component's unstability. A tool used for the creation of the unit tests is a Karma runner, which was developed by an AngularJS team. Under Karma there is Jasmine, which is a behavior-driven development framework for testing JavaScript code. An example of a unit test is shown and described in 4.1.

4.2 Functional tests

The main goal of functional testing is to verify whether the system possesses all the required functions. These tests confirm that the system performs the actions requested by the customer, apart from the unit tests which are a helpful mainly to the programmers [24].

As the main part of the component comprises visual elements which cannot be tested in any another way than by direct viewing, these tests were proceeded

manually in the browser.

```
1  it('dateFrom > dateTo', function () {
2
3    scope.signals = [{
4                      "_id":"573a41de274d3819362e7f11",
5                      "value1":887,
6                      "value2":778},
7                     {
8                      "_id":"573a41e8274d3819362e7f13",
9                      "value1":336,
10                     "value2":587},
11                    {
12                     "_id":"573a41ed274d3819362e7f14",
13                     "value1":493,
14                     "value2":650
15                    }];
16
17   var dateFrom = new Date (parseInt('573a41ed274d3819362e7f14'.
         substring(0,8), 16) * 1000);
18   var dateTo = new Date (parseInt('573a41de274d3819362e7f11'.
         substring(0,8), 16) * 1000);
19
20   expect(scope.updateGraph(dateFrom, dateTo)).toBe(false);
21 });
```

Listing 4.1: This test affirms that the function for rendering the customizable chart returns `false` in case the user picks a greater `dateFrom` than `dateTo`. The `scope.signals` array represents signals from the radiation monitor.

Conclusion

The aim of this chapter is to summarize the results achieved in this bachelor thesis and to introduce the future plans.

Result

The goal of this bachelor thesis is to create a web component enabling to visualise the data from a scientific instrument (neutron radiation monitor) in a real time. The analysis of the requirements that need to be fulfilled by this web component has been made in the first place as it was necessary for the designing all the component's parts. The implementation of the web component was proceeded based on the design of the component.

The result of the bachelor thesis is a web component which enables the real time data visualisation in a form of a chart. The operating data are simulating those, which will be later transmitted from neutron radiation monitor. The component consists of a real-time chart and a customizable chart displaying data based on the user's input in a form of picked time interval. On top of that the customizable chart is interactive as the user is able to zoom individual parts of it after picking the time interval. The component also offers the possibility of data retrieval in a graphics file format (only in Google Chrome) as well as in a text file format.

Future plans

The next step, which will be proceeded, is the deployment of the web component and the following connection to neutron radiation monitor, which will serve as the source of the data. Afterwards, the web component ought to get included into the website of the Van de Graaff accelerator, which is supposed to be developed in a few months.

Due to the fact that IEAP operates with a great deal of other data, which are retrieved from other scientific instruments in a similar way, it is potential to apply the concepts of this bachelor thesis to visualise other scientific data in the future.

Bibliography

[1] Nayak, A. *MongoDB Cookbook*. Packt Publishing, 2014, ISBN 1782161945, 9781782161943.

[2] Brantley, R. What Developers Mean When They Build a MEAN Stack [online]. 2014, [cit. 2016-22-04]. Available from: `https://www.newspindigital.com/nsd-tech-primer-the-mean-stack-a-k-a-full-stack-javascript/`

[3] Inc., G. Data Binding [online]. [cit. 2016-20-04]. Available from: `https://docs.angularjs.org/guide/databinding`

[4] Pasquali, S. Creating Real-time Applications with Node.js and Socket.IO [online]. 2015, [cit. 2016-25-04]. Available from: `https://masteringmean.com/lessons/398-Creating-Realtime-Applications-with-Nodejs-and-SocketIO`

[5] Granja, C.; et al. Basic and Applied Research with Light Ions and Tunable Mono-energetic Neutrons at the Prague Van-de-Graaff Accelerator. *Proc. Research and Applications of Accelerators Conference*, 2016.

[6] Roman, G. C. A Taxonomy of Current Issues in Requirements Engineering [online]. *Computer*, volume 18, no. 4, Apr. 1985: pp. 14–23, ISSN 0018-9162, [cit. 2016-21-04]. Available from: `http://dx.doi.org/10.1109/MC.1985.1662861`

[7] Bartholomew, D. SQL vs. NoSQL [online]. *Linux J.*, volume 2010, no. 195, July 2010, ISSN 1075-3583, [cit. 2016-21-04]. Available from: `http://dl.acm.org/citation.cfm?id=1883478.1883482`

[8] Parker, Z.; Poe, S.; Vrbsky, S. V. Comparing NoSQL MongoDB to an SQL DB [online]. In *Proceedings of the 51st ACM Southeast Conference*,

ACMSE '13, New York, NY, USA: ACM, 2013, ISBN 978-1-4503-1901-0, pp. 5:1–5:6, [cit. 2016-21-04]. Available from: http://doi.acm.org/10.1145/2498328.2500047

[9] MongoDB Inc. *MongoDB documentation [online]*. [cit. 2016-01-05]. Available from: https://docs.mongodb.com/manual/core

[10] W3Techs. Usage of server-side programming languages for websites [online]. Technical report, may 2016, [cit. 2016-18-04]. Available from: http://w3techs.com/technologies/overview/programming_language/all

[11] Mardan, A. PHP vs. Node.js [online]. 2013, [cit. 2016-18-04]. Available from: http://webapplog.com/php-vs-node-js/

[12] Sanchez, R. Comparing Node.js vs PHP Performance [online]. 2015, [cit. 2016-19-04]. Available from: http://www.hostingadvice.com/blog/comparing-node-js-vs-php-performance/

[13] Amico, C. Responsive Charts with D3 [online]. 2013, [cit. 2016-03-05]. Available from: http://eyeseast.github.io/visible-data/2013/08/28/responsive-charts-with-d3/

[14] Lubbers, P.; Greco, F. HTML5 WebSocket: A Quantum Leap in Scalability for the Web [online]. [cit. 2016-01-05]. Available from: http://www.websocket.org/quantum.html

[15] Richardson, L.; Ruby, S. *Restful Web Services*. O'Reilly, first edition, 2007, ISBN 9780596529260.

[16] Inc., M. MongoDB C Driver [online]. [software library]. version 1.3.5, [cit. 2016-01-05]. Available from: http://api.mongodb.com/c/1.3.5/

[17] Cadenhead, T. *Socket.IO Cookbook*. Packt Publishing, 2015, ISBN 1785880861, 9781785880865.

[18] Yeoman. AngularJS generator [online]. [software library]. version 1.8.1, [cit. 2016-01-05]. Available from: https://github.com/yeoman/generator-angular

[19] Lerner, A.; Powell, V. *D3 on AngularJS*. Leanpub, 2014.

[20] Bostock, M. Path Transitions [online]. 2012, [cit. 2016-02-05]. Available from: https://bost.ocks.org/mike/path/

[21] Bostock, M. Focus+Context via Brushing [online]. 2016, [cit. 2016-28-04]. Available from: https://bl.ocks.org/mbostock/1667367

[22] Koehler, W. Client-Side Solution For Downloading Highcharts Charts as Images [online]. 2014, [cit. 2016-014-05]. Available from: http://willkoehler.net/2014/11/07/client-side-solution-for-downloading-highcharts-charts-as-images.html

[23] Malmgren, D. example of how to export a png directly from an svg [online]. 2015, [cit. 2016-15-05]. Available from: http://bl.ocks.org/deanmalmgren/22d76b9c1f487ad1dde6

[24] Jorgensen, P. C. *Software Testing*. Boston, MA, USA: Auerbach Publications, 2007, ISBN 0849374758.

Appendix A

Acronyms

ACID Atomicity, Consistency, Isolation, Durability

API Application Programming Interface

BASE Basically Available, Soft-state, Eventual consistency

CORS Cross-Origin Resource Sharing

CRUD Create, Read, Update, Delete

CSS Cascading Style Sheets

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IEAP Institute of Experimental and Applied Physics

JS JavaScript

JSON Javascript Object Notation

LAMP Linux, Apache, MySQL, PHP/Perl/Python

MEAN MongoDB, Express.js, AngularJS, Node.js

MVC Model View Controller

ODM Object-Document mapping

ORM Object-Relational mapping

PNG Portable Network Graphics

REST Representational State Transfer

SOAP Simple Object Access Protocol

SPA Single-page application

SQL Structured Query Language

SVG Scalable Vector Graphics

URI Uniform Resource Identifier

VDG Van-de-Graaff accelerator

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... implementation sources
  - `github_link.txt` ..... link to GitHub repository
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format