Janne Kinnunen

Designing a Node.js full stack web application

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communication Technology

Bachelor’s Thesis

3 April 2023

Abstract

Author:	Janne Kinnunen
Title:	Designing a Node.js full stack web application
Number of Pages:	39 pages + 1 appendix
Date:	3 April 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication technology
Professional Major:	Smart IoT-systems
Supervisors:	Kimmo Sauren, Head of major

The aim of this thesis was to design and develop a full stack web application for an internet service provider, which can be used to manage and control its customers' internet connections.

The application was designed to be used on different devices, for example on mobile devices from the worksite, or on computer from the office. Bootstrap CSS framework which is designed for creating mobile-friendly websites was used for this purpose.

The application's server was built to a Node.js environment, which allowed for building a secure and versatile server thanks to its large ecosystem. In addition to the application's server, an API was built on to the same Node.js process, which the application uses to access the database. A separate program reads changes from the database and executes them on physical devices.

As a result, the desired features were implemented to the application, and it was launched on the company's intranet so that employees can use it.

This thesis examined the structure and operation of a full-stack web application, the requirements of the project and the selected technologies. The thesis discussed the architecture of the application, as well as the different stages of its development. Finally, possible future improvements and developments were discussed, such as new features or turning the user interface to a Progressive Web App.

Keywords: Full stack, web application, Node.js, API

Tiivistelmä

Tekijä:	Janne Kinnunen
Otsikko:	Node.js Full stack web sovelluksen suunnittelu
Sivumäärä:	38 sivua + 1 liite
Päivämäärä:	3.4.2023
Tutkinto:	Insinööri (AMK)
Tutkinto:	Tieto- ja viestintätekniikan tutkinto-ohjelma
Ammatillinen pääaine:	Älykkäät IoT -järjestelmät
Ohjaajat:	Kimmo Sauren, Smart IoT Systems tutkintovastaava

Tämän opinnäytetyön tavoitteena oli suunnitella ja kehittää internet palveluntarjoajalle full stack web sovellus, jota voidaan käyttää asiakkaiden internet-yhteyksien hallintaan ja valvontaan. Sovellus suunniteltiin käytettäväksi eri laitteilla, esimerkiksi mobiililaitteilla työmaalta tai tietokoneella toimistolta. Tätä varten käytettiin Bootstrap CSS -kehystä, joka on suunniteltu mobiiliystävällisten verkkosivustojen luomiseen.

Sovelluksen palvelin rakennettiin Node.js-ympäristöön, mikä mahdollisti turvallisen ja monipuolisen palvelimen rakentamisen Node.js:n suuren ekosysteemin ansiosta. Sovelluksen palvelimen lisäksi rakennettiin rajapinta samalle Node.js -prosessille, jonka kautta sovellus käyttää tietokantaa. Erillinen ohjelma lukee muutoksia tietokannasta ja suorittaa tarvittavat muutokset fyysisiin laitteisiin.

Lopputuloksena rakennettiin sovellus, johon lisättiin halutut ominaisuudet ja sovellus käynnistettiin yrityksen sisäverkossa, jolloin työntekijät pääsivät käyttämään sitä.

Tässä opinnäytetyössä tarkastellaan full stack web sovelluksen rakennetta ja toimintaa, projektin vaatimuksia ja valittuja teknologioita. Opinnäytetyössä käydään läpi sovelluksen arkkitehtuuri sekä kehityksen eri vaiheet. Lopuksi käsitellään mahdollisia tulevia kehityksiä, kuten uusia ominaisuuksia tai käyttöliittymän muuttamista progressiiviseksi web sovellukseksi.

Avainsanat: Full stack, web application, Node.js, API

Contents

1 Introduction

In the past, websites were content-based sites and they were meant to distribute information. However, as the internet has evolved, they have become a delivery platform for applications that used to be desktop applications or so called “web applications”. [1.]

Because web applications are used through a web browser, they can be used on different devices, such as computers or smart phones, if the web browser is compatible. Other benefits of a web application are the following:

- Every user has the same version of the application, meaning there are no compatibility issues.
- There is no need to install web applications to the device.
- Reduces the company’s costs because the application requires less maintenance and upkeep.
- Reduces the user’s costs by lowering the devices minimum system requirements.

[2.]

The goal of this thesis was to design and build a full-stack web application to Nivos Verkot Oy, so its employees can use the application to administer customers’ internet connections: change the connection speed, turn the connection on or off, and change other settings that help maintaining connections.

2 What is full stack web application?

Web application is a computer program that users can use through their web browsers. Full-stack applications have at least three main parts: User interface, called frontend, server, called backend and a database. Users use the frontend to send commands and requests to backend. Backend takes care of the cyber security, receives the commands and requests from user and uses the database. Figure 1 illustrates the structure of a full-stack web application. [3.]

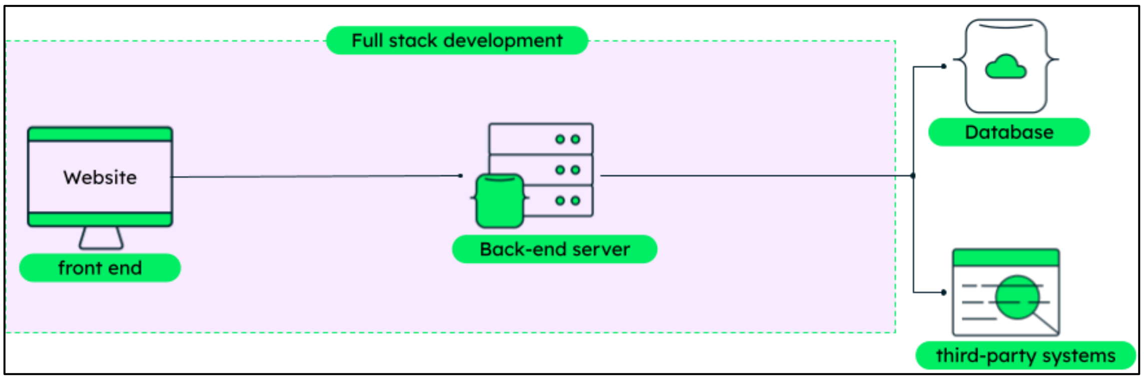


Figure 1. Structure of a full-stack system. [3.]

2.1 Frontend

Frontend means the user interface, in other words the part that user sees. A web application’s frontend is usually programmed using web browser-compliant programming languages, such as HTML, JavaScript, and CSS.

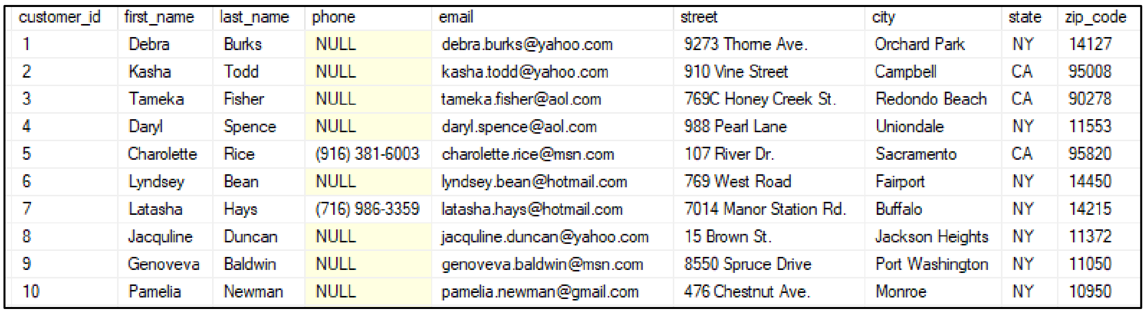
HTML is a markup language that web browsers can open and show to users. JavaScript code makes the frontend dynamic, for example, it will make buttons work. CSS is used to define the frontend’s appearance, such as colors, fonts or formatting. [1;4.]

2.2 Backend

When a user visits a website, the web browser sends a request to the web application’s server, also known as backend. Backend performs commands and requests that user sends, such as fetching data from database, processes the data and sends it back to user. User does not use backend directly but communicates with it using frontend. [1.] The backend is also responsible for the cyber security, performance and scalability of the web application.

2.3 Database

Database is a collection of organized data that is stored in a computer's memory. A Database Management System (DBMS) is an application that serves as a user interface between the user and the database. This allows users to fetch and update data and manage the way data is stored. The most common way of organizing data in a database is to arrange it into tables, rows and columns, in order to make data processing and usage as efficient as possible. [5.] Figure 2 illustrates a typical table of a database. This table holds different customer information, such as first and last name and contact information. Each row represents an individual customer, and the columns in that row describe their information.



customer_id	first_name	last_name	phone	email	street	city	state	zip_code
1	Debra	Burks	NULL	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14127
2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Beach	CA	90278
4	Daryl	Spence	NULL	daryl.spence@aol.com	988 Pearl Lane	Uniondale	NY	11553
5	Charolette	Rice	(916) 381-6003	charolette.rice@msn.com	107 River Dr.	Sacramento	CA	95820
6	Lyndsey	Bean	NULL	lyndsey.bean@hotmail.com	769 West Road	Fairport	NY	14450
7	Latasha	Hays	(716) 986-3359	latasha.hays@hotmail.com	7014 Manor Station Rd.	Buffalo	NY	14215
8	Jacquline	Duncan	NULL	jacquline.duncan@yahoo.com	15 Brown St.	Jackson Heights	NY	11372
9	Genoveva	Baldwin	NULL	genoveva.baldwin@msn.com	8550 Spruce Drive	Port Washington	NY	11050
10	Pamelia	Newman	NULL	pamelia.newman@gmail.com	476 Chestnut Ave.	Monroe	NY	10950

Figure 2. Typical table from a database [6.]

2.4 Application Programming Interface

Application Programming Interface (API) is used for communication between different software applications. Using an API, different software can communicate without needing to understand how each other one is programmed. This simplifies software development. [7.] APIs abstract away the more complex code and offer higher-level, simpler syntax instead. [8.] For example, software applications can fetch data from a database using an API.

3 Project requirements

The goal of this thesis project was to create a full-stack web application that employees can use to manage customers’ internet connections anywhere, from a work site via smart phone or from the office using a computer. The connection information is stored on a database, from which an external software reads changes and performs necessary operations, such as modifying the download speed setting of a physical end device.

The application will have multiple features, such as managing the settings of existing connections, such as download and upload speeds, activation, and deactivation, creating new connections, making mass changes to multiple connections and scheduling changes.

For accessing the database, a REST-API is created, which is an interface that allows different computer systems to transfer information and data over the internet.

Since users must be able to use the web application on smart phones or computers, the frontend must be responsive. This means that the user sees the same HTML file at the same URL address regardless of the user’s device, but the content may look different depending on the screen size. [9.]

The entire application should be secure, easy to maintain, require minimal maintenance and be easy to update in the future.

3.1 Node.js

Node.js is an open-source cross-platform JavaScript runtime environment. It is a popular tool for many different project and solutions, for an example, as an HTTP-server. One of its big advantages for frontend web developers is JavaScript programming language on the backend, removing the need to learn a new language. [10.]

Node.js application runs as a single process, so it does not create a new thread for each request. Node.js can perform a I/O operation, such as reading data from database, in two ways: blocking or non-blocking. Blocking way stops the Node.js process and waits for the data to be received. In non-blocking way, the process does not stop and wait, but the process returns to the operation when the data is received. This allows a Node.js process to handle thousands of requests and concurrent connections on a single server. [11.]

The JavaScript programming language has mechanisms, which can be used to divide the code to different modules that can be imported to the process if needed. [12.] This enables easier maintenance and readability of the code, as it can be divided into multiple smaller files instead of one large file. Node.js supports the CommonJS standard, which is a standard for adding modules primarily for servers. This standard does not work on web browsers, so JavaScript's own standard, ECMAScript, which aims to ensure website functionality across different devices and browsers [13], has created its own standard for importing modules. ECMAScript's standard, ESModules, works both on server and browsers, and is a modern approach to using modules.

Node.js has its own modules, that can, for example, create an HTTP server, read local files or even create child processes that can access the operating system in which the Node.js process runs. [14;15;16]
When Node.js is installed, a Node Package Manager (npm) is also installed. It is a package manager, which developers can use to download modules created by

other developers or upload their own modules. [17.] Npm is the largest software registry in world with more than two million packages. [18.]

Node.js was selected as a server-side environment for its flexibility, JavaScript programming language and its comprehensive ecosystem.

However, the modules that Node.js provided, does not provide all necessary features, additional modules must be added through npm. The selection of these modules must be based on their popularity, maintenance status and their publisher. This ensures that the selected modules are safe to use, and their life cycle is as long as possible, and they are actively maintained.

3.2 MariaDB

The Connection database has been built to a MariaDB -server before this project was started. The MariaDB -server runs in a company's Local Area Network, so the API must work within this environment. The API must integrate with the database as seamlessly as possible, allowing for any possible changes in database structure to be easily implemented into the API.

3.2.1 Data validation

The data in database must be in a correct format because a separate software reads data from the database and updates settings to physical devices. MariaDB provides the option to use enumerations as a datatype, which are predefined values. [19.] This way, the application can provide predefined options to user to choose from.

However, this option cannot be used in every case, as the value of the information may depend on the characteristics of the physical device. For an example, Nivos Verkot Oy provides internet connections to apartment building residents, but depending on the cabling of the building, the speed parameters may vary: If the building does not have CAT-cabling, fastest possible internet speed is 100/100M, but if the buildings with CAT-cabling, fastest possible speed might be as high as 1000/500M. [20;21.] This is the reason why enumerations cannot be used as a data format for all data in the database, as the value of the data may depend on the physical device's properties, because then the web application might offer same options for every connection. To overcome this problem, database stores information about device types and their supported parameters, such as speeds. The web application must check the suitable parameters based on the connection technology from the database and offer them to the user.

3.2.2 REST API

The API architecture used is Representational State Transfer (REST), which sets conditions on how the API should function. REST was originally created as a guideline for communicating in a complex network, such as the internet.

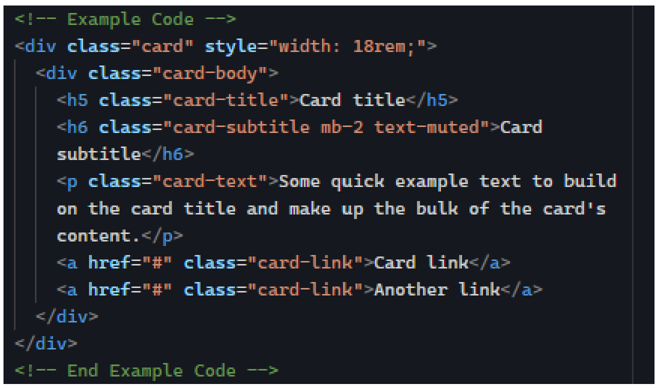
Basically, REST API's functionality is same as surfing in the web: User sends a request to the REST API's servers using HTTP protocol, and server returns the requested data to user in a predefined format. The request must include the following information:

- **Unique resource identifier (URI):** Server identifies every resource with a unique resource identifier.
- **Method**: REST-APIs typically operate using the Hypertext Transfer Protocol (HTTP). The HTTP method informs the server what it needs to do to the resource. There are several methods available, with the most common ones being:
    - **GET**, which is used by the user to request information.
    - **POST**, which allows the user to send new data to the server.
    - **PUT**, which enables the user to update existing information.
    - **DELETE**, which is used by the user to request the server to remove information.

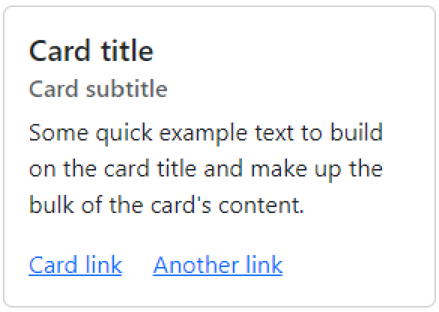
[22.]
Because REST-API works on the same HTTP-protocol as a web application, the API can be built to a Node.js -process. MariaDB provides its own npm -package for Node.js, making it easy and secure to use the database. [23.]

3.3 Bootstrap

Responsiveness of websites plays a significant role in today's world as an increasing number of people use devices other than computers to browse websites. [9.] The responsivity can be achieved by using CSS code, which modifies the layout of a website, depending on what size is the screen of the user's device. [4.] Due to the increasing need for websites to be mobile-friendly, there are CSS frameworks available that help creating responsive websites. In this project Bootstrap framework is used, which is one of the most popular CSS frameworks. It is designed to build mobile-first websites. [24.] One of the advantages of a CSS framework is that it is easy and quick use in a project. Instead of creating a separate file that refers to the HTML-elements, the framework provides predefined commands that can be directly written into the HTML-elements' class attribute to define their appearance, as shown in figure 3, Bootstrap provides components and features, which can be used to build different layouts. In figure 3, Bootstrap's "card" feature is used to build a card to a website.



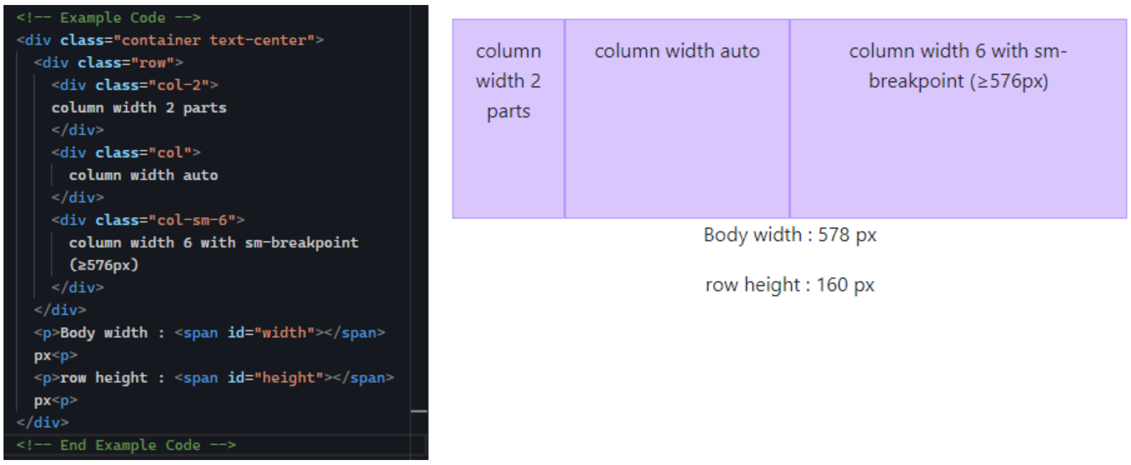
```html
<!-- Example Code -->
<div class="card" style="width: 18rem;">
  <div class="card-body">
    <h5 class="card-title">Card title</h5>
    <h6 class="card-subtitle mb-2 text-muted">Card
    subtitle</h6>
    <p class="card-text">Some quick example text to build
    on the card title and make up the bulk of the card's
    content.</p>
    <a href="#" class="card-link">Card link</a>
    <a href="#" class="card-link">Another link</a>
  </div>
</div>
<!-- End Example Code -->
```



Card title
Card subtitle
Some quick example text to build on the card title and make up the bulk of the card's content.
Card link Another link

Figure 3. Bootstrap's example how "card" feature is used to build a card. [25.]

Bootstrap provides a grid system, that can be used to divide the website into a grid. In the grid system, webpage is divided to rows and columns, with the content is within the columns. Each row is divided to 12 parts, where one column's width can be at least one part, but at most 12 parts. Each row can hold at most 12 columns if every column's width is one part. It is possible to assign each column its own width, but if it is not assigned, then Bootstrap will automatically set the width of column. [26.] In figure 4, an example row has been built using Bootstrap's grid system. Notice that there is a div element with class "row" that and inside that element there are three div elements that have classes "col". Columns width can be assigned by adding a number from 1 to 12 after the "col" class attribute. The width of the first "col" element has been set to two parts, the width of the second element is determined by Bootstrap, and the width of the third "col" element is six, which has also been assigned an sm-breakpoint.

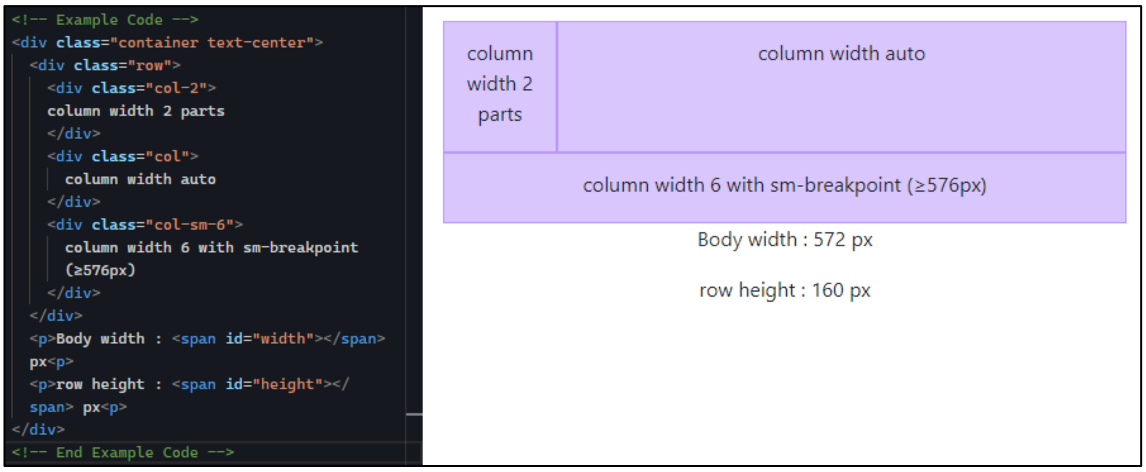


```html
<!-- Example Code -->
<div class="container text-center">
  <div class="row">
    <div class="col-2">
    column width 2 parts
    </div>
    <div class="col">
      column width auto
    </div>
    <div class="col-sm-6">
      column width 6 with sm-breakpoint
      (≥576px)
    </div>
  </div>
  <p>Body width : <span id="width"></span>
  px<p>
  <p>row height : <span id="height"></span>
  px<p>
</div>
<!-- End Example Code -->
```

Figure 4: Example code of a Bootstrap row.

Bootstrap offers breakpoints, that define how the appearance of a webpage behaves on different screens. [27.] Breakpoint is a minimum width defined by Bootstrap, that can be assigned to elements, for an example, columns. If the width of the device's screen is bigger than the breakpoint, the column remains the same size as defined, for example, taking up three parts of the row. If the device's screen is smaller than the breakpoint, then the width of the column is 12 parts. [26.] Bootstrap's grid system will arrange the remaining columns evenly.

The example in figure 5 has the same source code as figure 4, but the width of the screen is 572 pixels instead of 578 pixels. The column that had the breakpoint assigned, is now 12 parts wide. The column that had a width of two parts, will remain at the same size. The middle column that had no assigned width, fills the remaining space of the row. Note that the overall row height remains the same, so the Bootstrap's grid system adjusts the column heights as needed.



```
<!-- Example Code -->
<div class="container text-center">
  <div class="row">
    <div class="col-2">
    column width 2 parts
    </div>
    <div class="col">
      column width auto
    </div>
    <div class="col-sm-6">
      column width 6 with sm-breakpoint
      (≥576px)
    </div>
  </div>
  <p>Body width : <span id="width"></span>
  px<p>
  <p>row height : <span id="height"></
  span> px<p>
</div>
<!-- End Example Code -->
```

column width 2 parts

column width auto

column width 6 with sm-breakpoint (≥576px)

Body width : 572 px

row height : 160 px

Figure 5. Same example code with Figure 4, but the screen size is reduced.

3.4 Information security

In today's world, information security plays a crucial role by protecting files and software from unauthorized and malicious use. [28.]

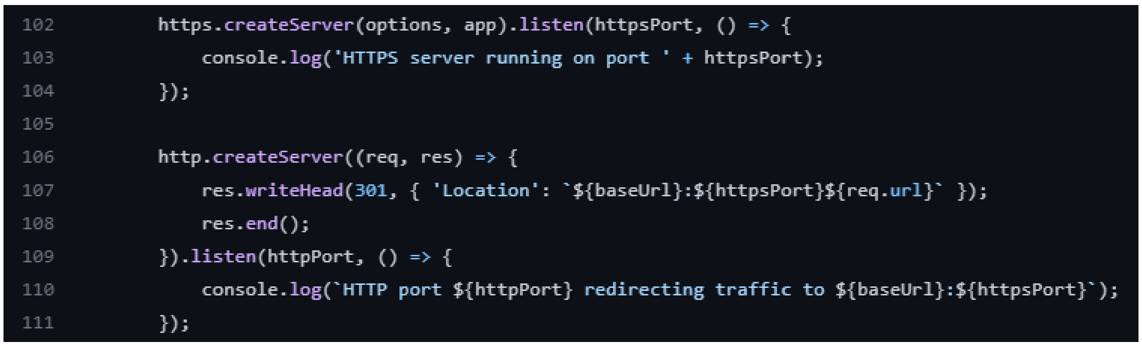
3.4.1 Client-server communication

HTTP protocol is used for communication between a website and a server, and most of the data transmission on the internet uses this. Downside of this protocol is that data transmissions can read by anyone who is monitoring the session, because this protocol is sending plain text data. To prevent this, HTTPS protocol is used instead of HTTP. HTTPS is almost the same as HTTP, but it uses TLS/SSL method to encrypt normal, plain text HTTP messages. TSL/SSL method uses public key cryptography: Two keys are generated, public and private. Public key is shared with the client. When client connects to the server, both server and client uses both keys to create a new key, session key, that will be used to encrypt messages. [29.] This protocol must be used both in the API and in the web application.

When a user wants to browse websites, user must know the address of the server and a port number. Port is an application-specific software construct, that works as a communication end point. A server must be configured with a specific port that it will use to listen or incoming connections.

In 1991, Tim Berners-Lee defined that “if the port number is not specified, 80 is always assumed for HTTP.” [30.] This is the reason why users don’t have to know the port number, because browsers can send the data to port 80 by default. Because the communication between the client and the server needs to use HTTPS protocol, port 80 cannot be used. HTTPS has a default port number, 443, which has been defined in year 1994. [31.] To address this issue, two servers must be created to the Node.js process. One server listens to the port 443 and second listens to port 80. The purpose of the second port is only to redirect users to the first server, that will work as the main server for this project. This way client-server communication uses HTTPS -protocol and the data transmission is secured.

Figure 6 illustrates the creation of these two servers. On rows 102-104, server that listens to port 443 is created with parameters of options and app. Options -parameter includes SSL keys, and app parameter is the express-framework that the server will use for every request. On rows 106-111, server that listen to port 80 is created. Its only purpose is to redirect requests to the other server. Note that at row 107, HTTP status code 301 is given, so the client knows to communicate to the port 443 in the future.

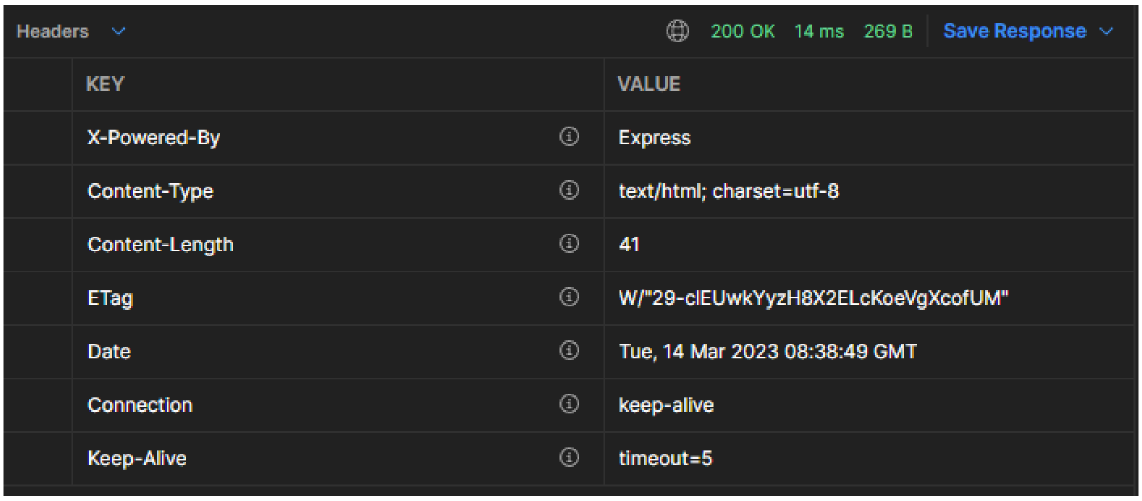


```
        https.createServer(options, app).listen(httpsPort, () => {
            console.log('HTTPS server running on port ' + httpsPort);
        });

        http.createServer((req, res) => {
            res.writeHead(301, { 'Location': `${baseUrl}:${httpsPort}${req.url}` });
            res.end();
        }).listen(httpPort, () => {
            console.log(`HTTP port ${httpPort} redirecting traffic to ${baseUrl}:${httpsPort}`);
        });
```

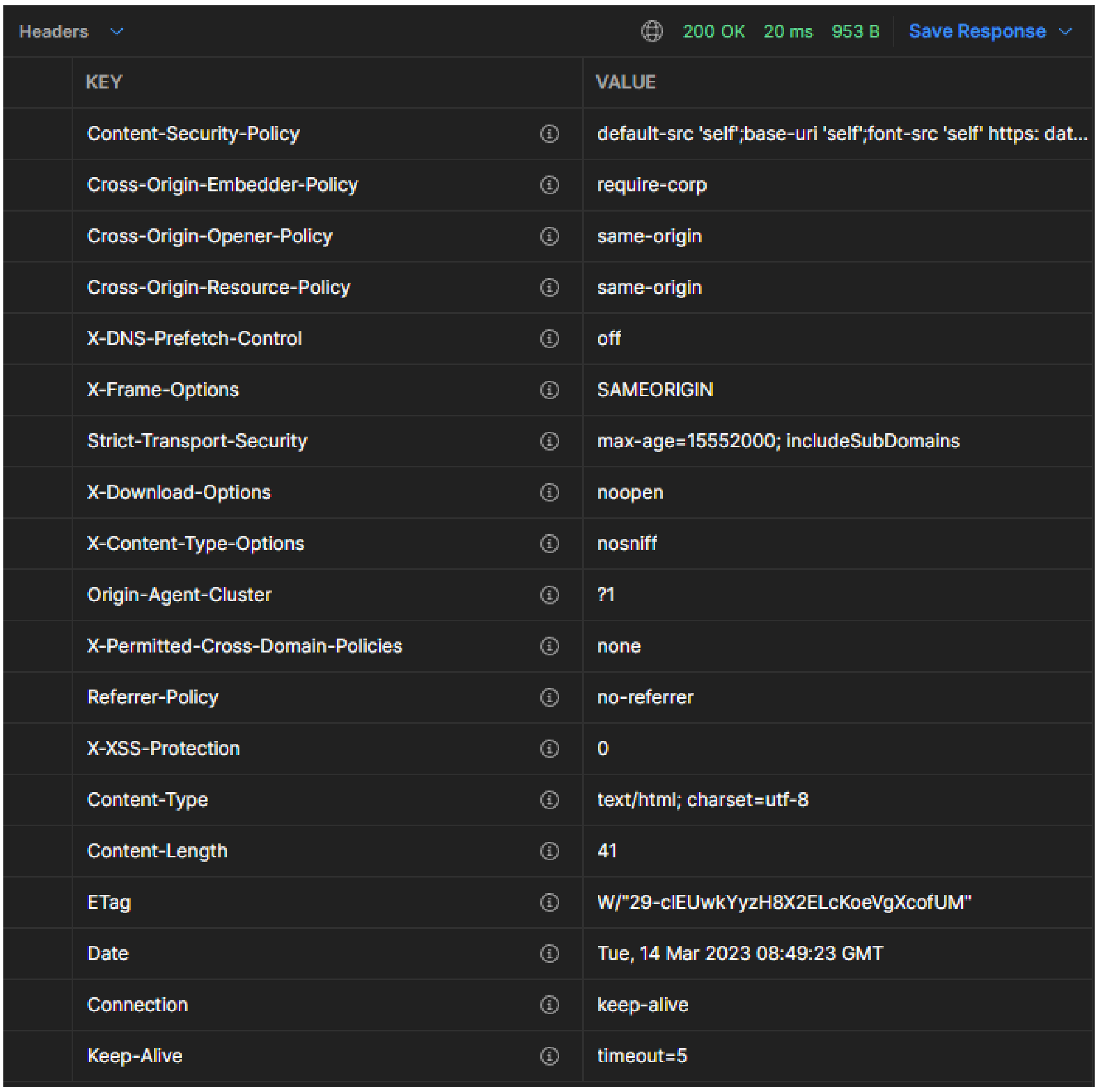
Figure 6. Creating two servers to node.js process

Usually, a server adds headers to HTTP protocol, such as the X-Powered-By header, which describes what technologies the server is using. When using Express framework, it adds “Express” to this by default. This information can reveal to potential attackers what server is used, which they can use to search for vulnerabilities in the server. [31.] Npm package called Helmet has been created to hide this information. Using Helmet, headers can be hidden or modified, making it more difficult for attackers to determine which server is being used. [32.] Figure 7 shows what are the basic headers that Express sets, and in Figure 8 are the headers that Helmet sets by default.



KEY	VALUE
X-Powered-By	Express
Content-Type	text/html; charset=utf-8
Content-Length	41
ETag	W/"29-clEUwkYyzH8X2ELcKoeVgXcofUM"
Date	Tue, 14 Mar 2023 08:38:49 GMT
Connection	keep-alive
Keep-Alive	timeout=5

Figure 7. Default headers that Express -framework sets.



KEY	VALUE
Content-Security-Policy	default-src 'self';base-uri 'self';font-src 'self' https: dat...
Cross-Origin-Embedder-Policy	require-corp
Cross-Origin-Opener-Policy	same-origin
Cross-Origin-Resource-Policy	same-origin
X-DNS-Prefetch-Control	off
X-Frame-Options	SAMEORIGIN
Strict-Transport-Security	max-age=15552000; includeSubDomains
X-Download-Options	noopen
X-Content-Type-Options	nosniff
Origin-Agent-Cluster	?1
X-Permitted-Cross-Domain-Policies	none
Referrer-Policy	no-referrer
X-XSS-Protection	0
Content-Type	text/html; charset=utf-8
Content-Length	41
ETag	W/"29-clEUwkYyzH8X2ELcKoeVgXcofUM"
Date	Tue, 14 Mar 2023 08:49:23 GMT
Connection	keep-alive
Keep-Alive	timeout=5

Figure 8. Default headers that Helmet sets.

3.4.2 Client authentication

Users must be logged in before they can use the web application. To create a secure way to authenticate users, Passport is used, which is an authentication middleware that is directly compatible with Express middleware. Passport is a popular middleware, with a weekly download count of 2 068 408 as of March 22, 2023. [33.]

The purpose of passport is to authenticate user. Passport provides many different strategies for this, 538 different strategies as of March 22, 2023. Different strategies that Passport provides, are for example, Facebook-, Google- or Twitter authentications, or Local strategy, where username and passport has been stored, as the strategy name suggests, locally. This is a popular option, as the time of writing on March 27, 2023, it has been downloaded 603 610 times in the last week. [34.] This strategy is suitable for the project, as it makes it easy for the company to manage users.

To make the authentication more secure, passwords must be encrypted before storing them. If an unauthorized person gains access to user credential, they will not obtain the plaintext password but a protected version that cannot be used to logged in. [37.] To encrypt passwords, the Bcrypt npm package is used, which provides tools for this purpose. [37.]

HTTP protocol is stateless, meaning that every request that client sends, can be treated as a separate message – with no knowledge of last requests or messages. This presents a problem for web applications because it requires users to log in between every request. [38.]

To solve this problem, sessions must be created between the client and the server. When user logs into the application, the server creates a session and an HTTP cookie, which contains the session data. Then this cookie is sent to the user, and the user will send it back to the server with every request. The server can use this cookie to identify the session and the user. This way a stateful protocol can be created on top of HTTP protocol. [38.] Express middleware has an additional npm package, express-session, that will also work with Passport. [38.]

In addition, users are assigned their own roles, which can be used to restrict the access to resources and features. This can be used to give, for example, customer service only read access or limited editing rights, such as increasing speed or disconnecting a connection.

3.4.3 API security

Just like the web application, API must be protected from malicious use. The interface is stateless, as information is only requested from it when needed, so there is no need for the sessions that are used in web application.

To protect the API, a JSON Web Token (JWT) method is used. JWT is an independent way to securely transfer information in JSON format, such as identification data. This information can be verified as it is digitally signed. The JSON Web Token can also have an expiration time, for example, one hour, during which time Token will work. After this API stops approving this token.

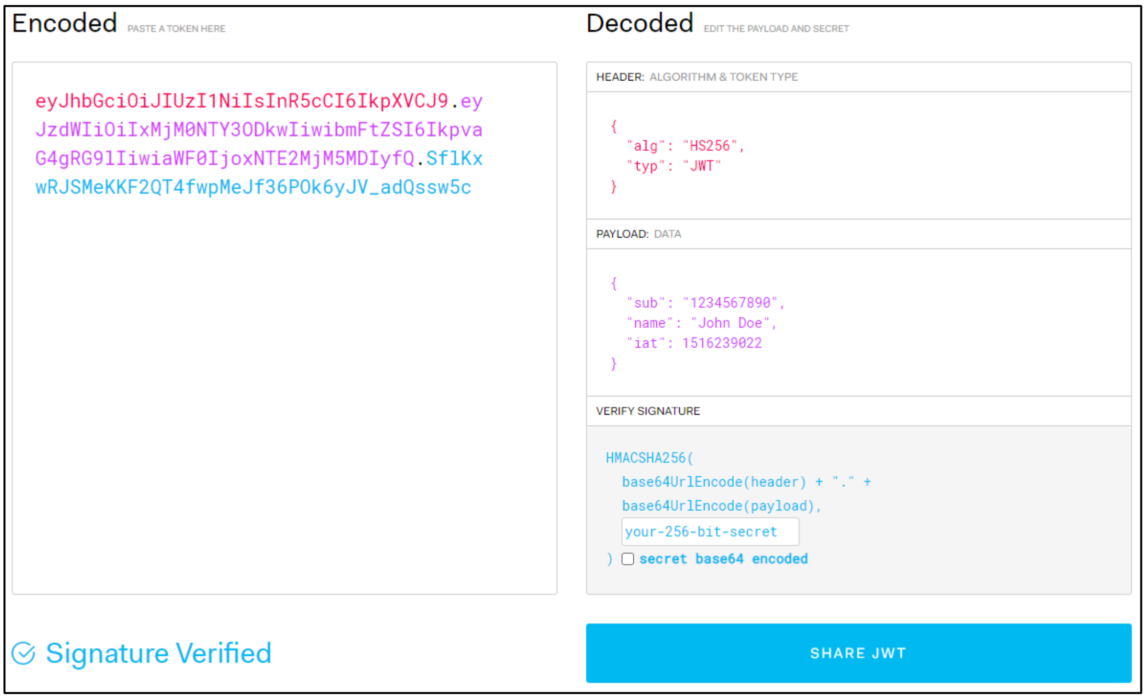
JWT is an encrypted string, which contains three parts:

Header
Header is built from two parts, type of the token, which is JWT in this case, and a signature algorithm that will be used.

Payload
Payload contains the information that will be transferred, such as user information and expiration time.

Signature
Signature is used to make sure that the message has not changed. To create a signature, a string must be created that holds the header, payload, and a secret key, which will be encrypted with an algorithm that is mentioned in header.

Figure 9 illustrates what information JWT can hold and what it looks like encrypted:


Encoded PASTE A TOKEN HERE
Decoded EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
PAYLOAD: DATA
VERIFY SIGNATURE
secret base64 encoded
Signature Verified
SHARE JWT

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
)
```

Figure 9. Encoded and decoded JSON Web Token. [39]

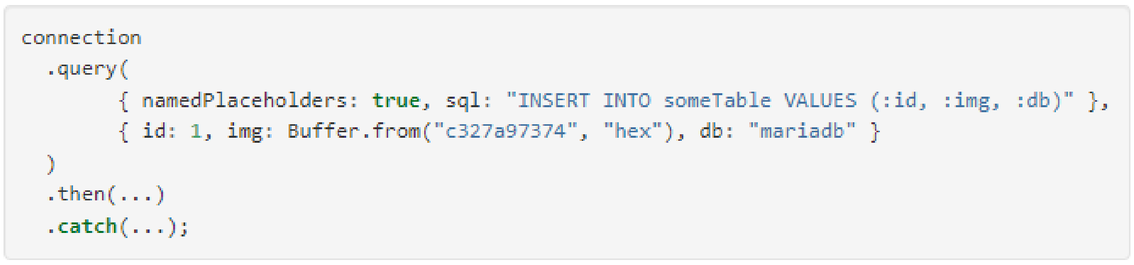
When user logs into the web application and requests data from server, server authenticates user, uses the JWT that is created for the user, and sends it to API in a HTTP Authorization Header. Then API checks the JWT and makes sure that the user has access to the resources they have asked.

Using this method API is not limited only for the web application, but JSON Web Tokens can be created for other software applications that intended to use the API.

One of the most common hacking techniques is SQL injection, where SQL commands are added to the normal user input. This way a hacker can have sensitive data from database, change it or even remove data. SQL injection is possible, if API adds user given parameters straight to the SQL command it intends to execute. [40.]

By default, hackers cannot use API if they do not have a valid JSON Web Token, as mentioned previously. However, it is still a good practice to add protection against SQL injection. The MariaDB npm package offers a ready-made solution for this: this package only allows one SQL command to be executed at a time and parameters can be added as placeholders to the command. This way, the MariaDB package can verify that the parameters are in the correct format and are not harmful.

Figure 10 shows an example from MariaDB’s own website, how to add parameters to the SQL command using placeholders. In the figure, connection.query() function executes SQL command and returns the received data. Function can take parameters in the form of JSON objects that are common in JavaScript. The first parameter contains the SQL command with the placeholders :id, :img and :db added to it for the parameters. The second JSON object parameter contains the parameters entered by the user. [41.]



```
connection
  .query(
        { namedPlaceholders: true, sql: "INSERT INTO someTable VALUES (:id, :img, :db)" },
        { id: 1, img: Buffer.from("c327a97374", "hex"), db: "mariadb" }
  )
  .then(...)
  .catch(...);
```

Figure 10. Example from MariaDB on how to add parameters to SQL command.

Cross-site scripting is a vulnerability, that hackers can use to inject malicious scripts to website, that gets executed when other users visit the website. These scripts can steal sensitive information, such as credentials, or they can modify website’s information. Common way for hackers to perform this, is to save scripts into database through normal user input. Then, when unsuspecting user visits the website, the malicious script is loaded from the database and executed. [42.]

This is not a danger in this project, as the web application is limited only to specifically designated employees. However, Helmet npm package provides a security layer that helps to identify and prevent certain types of attacks, such as Cross-site scripting and data injections. [32;43.]

4 Application structure

To make the source code of the application as easy to maintain as possible, its structure must be easy to understand. Therefore, the structure of the software must be carefully planned.

4.1 Backend structure

The API and the backend of the web application are built to the same Node.js process, since the web application is the only software that will use the API and the API is mainly built for the web application. This simplifies the development work and software management since this removes the need for doing changes to multiple different software.

Even though API is in the same process as the web application, web application’s backend uses API to fetch data. This way information security is simplified, as it is the only way to access the database, and also makes it easier to separate API to an independent process, if needed.

The chosen architecture for the web application’s backend follows a practice similar to the MVC architecture, which consists of the Model, View and Controller components. [44.] Figure 11 illustrates how the MVC architecture works.

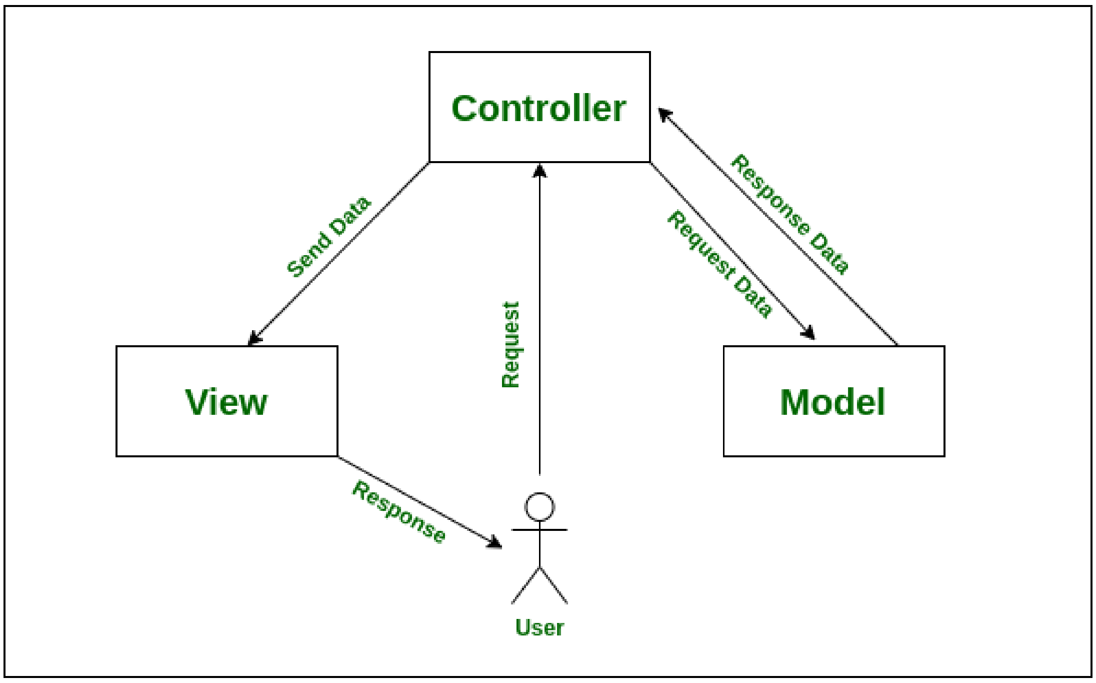
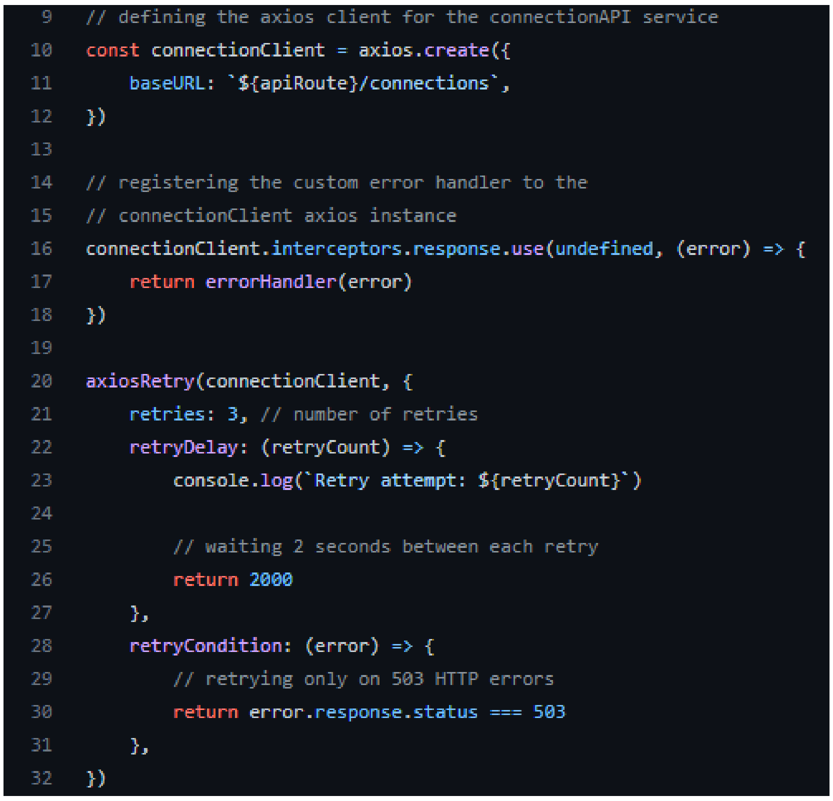


Figure 11. MVC architecture [45.]

Model

The role of the model is to handle the data processing. Because data is fetched from API, the model does not use the database but instead uses the Node.js' Axios module to make HTTP requests to API.

The Axios module is used to create instances for all the data in the database, such as connections, apartment buildings or fiber optic terminal devices. Each instance is defined with functions that will send requests to the API. Figure 12 illustrates the creation of the connection client that is created using Axios. These instances can have their own functions that will send the requests to the API.



```
// defining the axios client for the connectionAPI service
const connectionClient = axios.create({
    baseURL: `${apiRoute}/connections`,
})

// registering the custom error handler to the
// connectionClient axios instance
connectionClient.interceptors.response.use(undefined, (error) => {
    return errorHandler(error)
})

axiosRetry(connectionClient, {
    retries: 3, // number of retries
    retryDelay: (retryCount) => {
        console.log(`Retry attempt: ${retryCount}`)

        // waiting 2 seconds between each retry
        return 2000
    },
    retryCondition: (error) => {
        // retrying only on 503 HTTP errors
        return error.response.status === 503
    },
})
```

Figure 12. Creating a connection client using Axios.

View

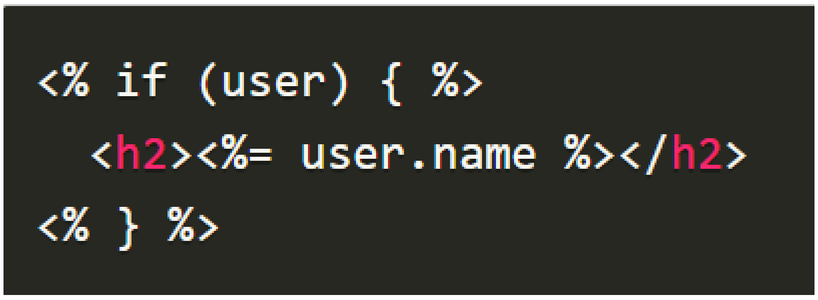
The role of the view is to handle presentation layer, which means modifying the page that the user sees as needed.

There are different ways to implement the View’s task, such as client-side rendering, where user’s device receives the requested data and dynamically modifies the HTML file to display the data. This method reduces server load and allows for a more dynamic user interface but can slow down the loading especially in slow networks and requires additional programming to create functions that receive and organize data into the HTML file. [46.]

Another method is server-side rendering, where the server adds user requested data to HTML file before server sends it to the user. This way, only the desired information can be displayed to the user, without sending any unnecessary information, for example, creation time of the connection and other metadata, which is returned by the API. This approach is faster for the initial load and reduces the required performance of the user’s device. However, this method requires more resources from the server. In addition, creating dynamic pages is more difficult because the server returns a new HTML file instead of just the desired information. [46.]

In this project, server-side rendering is used, since it reduces the amount of programming and it increases the information security, because user’s device does not communicate with the API, and only the desired information is added to the HTML file, that is then sent to the user.

To implement server-side rendering, EJS (embedded JavaScript) module is used. EJS is a simple template language, which can generate a HTML file using JavaScript. HTML code can be written in an EJS file and JavaScript can be used in necessary places with tags. [47] Because of this, it has a low learning curve, making it easier to make modifications to the software. Figure 13 illustrates how EJS tags work: JavaScript can be written inside the “<%” and “%>” tags on the first line, for example to check if the “user” variable exists. This line is not shown for users, because “<%” tag is so called scriptlet tag, which does not produce text for the client. On the second line, the JavaScript code written after “<%=” tag, will produce text to the client. In this example, the user.name variable’s data is written to the HTML file. On the last row, the JavaScript code between the scriptlet tag closes the if -statement. This method allows the middle line to be rendered to the HTML file only if the “user” variable has been defined during the rendering phase. [47]



```
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

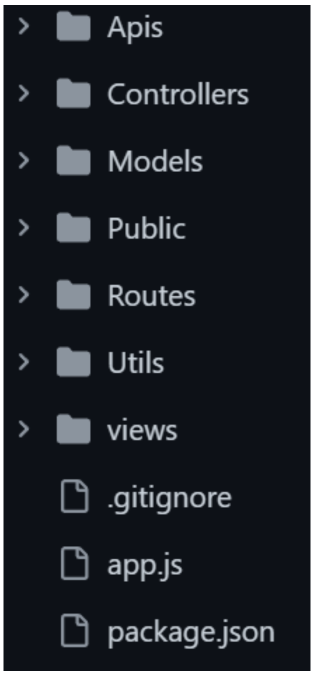
Figure 13. Example use of the EJS tags.

Controller

The purpose of the Controller is to work between the view and the model. It receives the user requests, asks data from the model, and provides the data to view. [44] Because model returns all the requested data from API, the controller can handle this data and select what is sent to the view.

Managing the codebase

To keep the management and maintenance as easy as possible, code files are divided into directories and subdirectories. This makes it easy to navigate to a specific part of the software. Figure 14 illustrates the files and folders that the project requires. Every directory and file are not accessible for the normal application user, except the Public directory. This directory contains the necessary CSS and JavaScript files that the frontend requires. The app.js file works as a ”core” for the application. It contains the code that configures and starts all the necessary parts of the process, such as connecting to the database and opening a HTTPS-server.



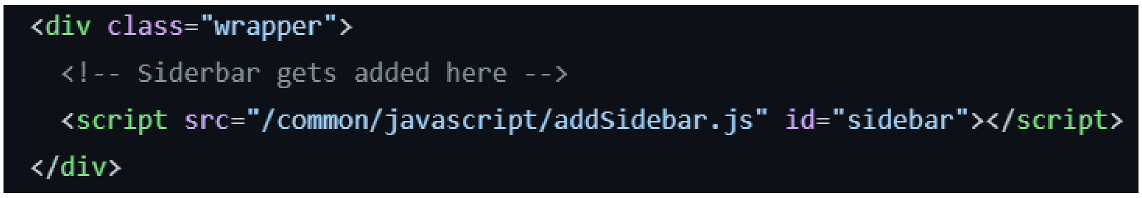
```
> Apis
> Controllers
> Models
> Public
> Routes
> Utils
> views
  .gitignore
  app.js
  package.json
```

Figure 14. Directories of the project

4.2 Frontend structure

Because the web application has multiple different features, such as modifying connection, creating new connections, or performing mass changes to connections, every feature needs a separate .EJS file, which the server can use to render all necessary data.

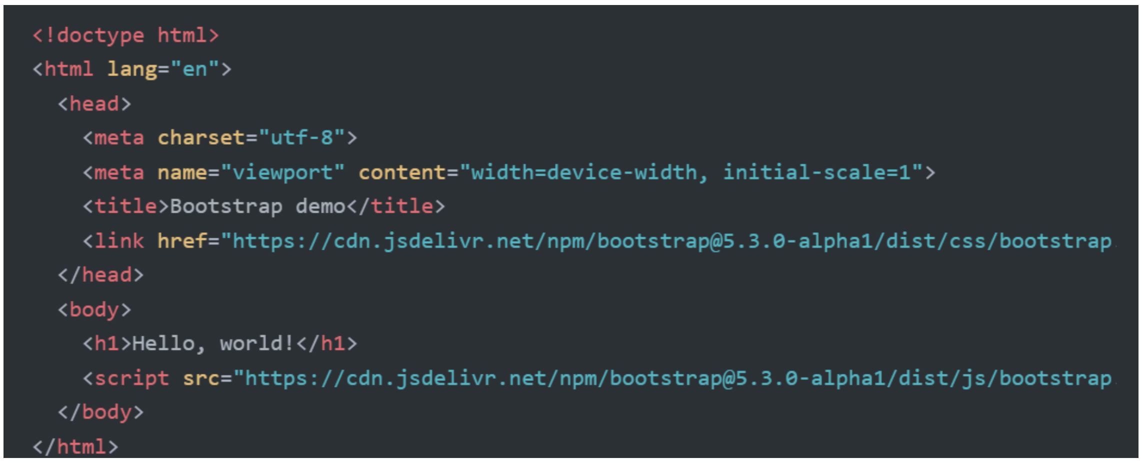
Because of the multiple features, a navigation bar is needed so user can navigate between different features. This navigation bar needs to be added to every .EJS file. This creates a problem: If the navigation bar needs an update, the update needs to be done for every .EJS file, to keep the navigation bar consistent. This problem is solved by using a Partial View method, where large HTML styled markup files are divided into smaller components. This makes construction and maintenance of the frontend easier. [48.] In this project, a separate HTML file is created for the navigation bar, which contains only the navigation bar elements. Then this HTML file is sent to the user's device inside the Public -directory with a separate JavaScript code, that will attach the navigation bar to the HTML file at the frontend. With this method, there is need for only one navigation bar file and the JavaScript file can be added as a link to the .EJS file, like shown in Figure 15.



```
<div class="wrapper">
  <!-- Siderbar gets added here -->
  <script src="/common/javascript/addSidebar.js" id="sidebar"></script>
</div>
```

Figure 15. Link for a JavaScript file that attaches sidebar to the HTML file in client side.

For making the web application responsive, Bootstrap framework is used. Bootstrap can be added to the project by installing it as a npm package or using the Content Delivery Network (CDN) approach. CDN is a geographically distributed group of servers, from which users can download files. [49.] Using CDN, there is no need to install any Bootstrap files into the project. By adding a link to the HTML file, user's device can load necessary files and script using the Content Delivery Network [49.], like shown in Figure 16. This also makes it possible to always have the latest version of Bootstrap available to the user.



```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Bootstrap demo</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap
  </body>
</html>
```

Figure 16. Bootstrap's example for using CDN approach. [50.]

4.3 API Structure

A similar structure to the backend is created for API, with the difference that there is no Views section. The API's Controller receives requests and uses Model that retrieves requested data from database and then returns it to the Controller. Then Controller sends the retrieved data as a HTTP response to the sender of the HTTP request.

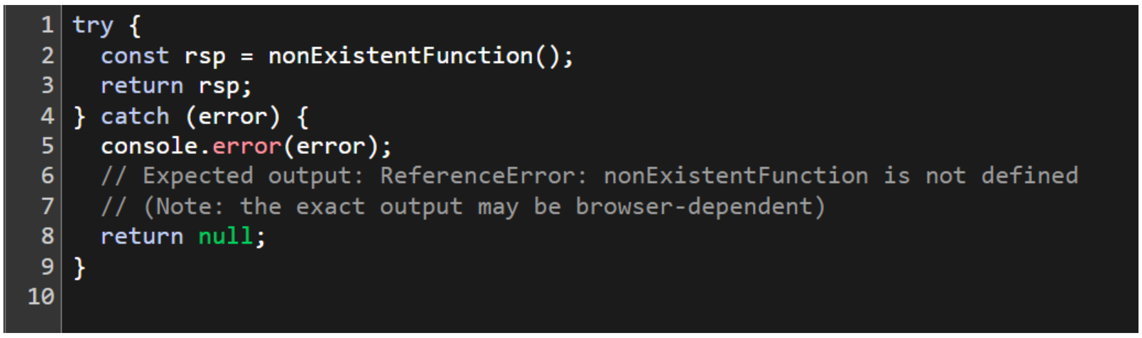
MariaDB's npm package is used to read the database, and a custom Model is built for the API. Using this Model, API can access the database, such as reading and modifying data, or even get information about the structure of the database.

To ensure that the API works as seamlessly with the database as possible, the API does not hard code the names of the database columns in SQL commands. Instead, the user must input the column names and values to the request. Before creating a SQL command, API fetches the real column names from the database and compares these names to the ones that user has input. If column names match, then API adds that parameter to the SQL command. This ensures that no invalid SQL command is built. As a result, any potential changes to the database, such as adding or removing a column, will not necessarily require any modifications to the API software.

4.4 Error handling

One downside of the Node.js process is that if an error occurs, the whole process shuts down. Therefore, it is important to design the error handling carefully. Errors could occur for example, faulty code, API returns something unexpected, the process cannot connect to API, or some file is missing.

One way to handle the errors is to use JavaScript’s throw mechanism, which is used in a “try..catch” structure. [51.] The use of a throw mechanism generates an exception, that must handle using try..catch structure, or Node.js process shuts down immediately. [52.] In Figure 17 is illustrated, how try..catch structure is built. When the example program is trying to execute the function called in line 2, an exception occurs and program jumps over to line 4, after which the program executes lines 5-9.



```
 1 try {
 2   const rsp = nonExistentFunction();
 3   return rsp;
 4 } catch (error) {
 5   console.error(error);
 6   // Expected output: ReferenceError: nonExistentFunction is not defined
 7   // (Note: the exact output may be browser-dependent)
 8   return null;
 9 }
10
```

Figure 16. Example of a try..catch structure

In both API and web application’s backend, the try..catch structure is built to the Controller to handle errors that occur somewhere later in the process that user has started. An error handler -functions are created for both parts, the purpose of these functions is to additional information, like the HTTP status, to the error information, to modify error information to a user-friendly form, and to send the error information to the user. The API’s error function sends the error information in a JSON -format, whereas the web application’s backend renders the error information to a .EJS file and sends that to user.

The try..catch structure can handle errors created by the software, but also custom errors defined by the developer. For an example, the API can check the request from user and if this request does not contain all the necessary information, or user does not have the rights, then a custom error can be thrown with custom messages. Then this error can be handled in a same way as the default errors are handled. [53.]

5 Project Roadmap stages and tools

5.1 Tools

To identify and track different stages of the project, Miro, a browser-based whiteboard platform was used alongside with Microsoft Word. The roadmap of the project was created in Miro, which helped to organize different parts of the project and enable tracking of the required steps. (Appendix 1)

Microsoft Word was used to record in detail the different features what the project has, and in what stages the features are. Additionally, a journal was maintained in Word, documenting what was achieved during the day, identifying any issues that arose, and highlighting what needed to be clarified to solve these issues.

Regular meetings were not held during the project, but instead meetings were arranged if there was need to discuss about the features or issues.

For programming, Visual Studio Code was used, which is a free lightweight source code editor provided by Microsoft. [54.] It has an integrated terminal [55.], which is a convenient way to initiate, debug, monitor and terminate

Node.js processes during development. Npm packages can also be installed using this terminal.

Another tool available for development is Nodemon, which can be installed from npm. Nodemon makes development of Node.js processes easier. Using Nodemon, Node.js process restarts automatically, when file changes in the directory. [56.] This makes it easier and faster to test Node.js processes.

For saving the codebase to cloud and controlling versions, GitHub was used, which is a code hosting platform for version control and collaboration. [57.] GitHub can be used, among other things, to review changes made to the project.

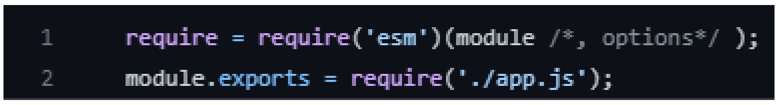
For testing the API, an application called Postman is used. This application allows customized HTTP requests to be send to the API, and its functionality to be checked. This makes it easier and faster to test and develop API. [58.]

5.2 Roadmap stages

The stages of the project have been divided into main headings in roadmap to keep the overall picture clear. In addition, the biggest stages in roadmap can be opened in Miro and write more detailed information about what needs to be done.

In the initial stage of the work, the necessary directories are created for the application, but the files within the directories are created as the work progresses. In addition to the directories, a package.json file is created, that contains metadata about the project, such as project name, version, starting commands, license and installed modules and packages, which are also referred to as dependencies.

By default, Node.js does not support the ESModules standard, it must be enabled by installing and using an esm package from npm. After installing, Node.js process can be started by adding a parameter “esm” to the starting command. For example, the normal starting command is “node app.js”, so using “node esm app.js” will enable ECMAScript standard. To simplify the start and use of the process, a separate starting script will be created, that loads and enables the esm module before starting the app. [59.] This way the process can be started with the normal starting command, but instead using app.js, the starting script, start.js is used. Figure 17 shows what happens inside the starting script.



```
require = require('esm')(module /*, options*/ );
module.exports = require('./app.js');
```

Figure 17. start.js script

After these initial stages, the application can be constructed according to the order planned in the roadmap.

At the beginning of the project, express server is built, which works as a base for whole application. In this stage, the initial settings and configurations are set to the express, for example, which directory can be shared to the client and in what format client will send data. At this stage, Helmet package is also initialized.

Next step involves creating a login system and establishing a connection to the database. By creating the login system at this stage, user authorization can be added to all necessary parts of the project during the development, eliminating the need to add them later.

The biggest stage is to build the web application and API. This approach allows for a better understanding of the requirements that web application needs from the API. For example, to search connections by address, the SQL command needs to have a “LIKE” clause with a wildcard for searching all connection in the same building, instead of searching only for the specific apartment in the building. However, the use of this feature should be limited to address parameter searches only, so the API needs to recognize when a user is searching by address to activate this feature.

When web application and API is ready, the JWT authentication is created. This is done only after the API is completed, to speed up the process of building and testing it, as there is no need to login and create a JWT every time API is tested.

When the application is deployed to production, TLS encryption is enabled. To enable encryption, SSL certificates are required, which contain information such as the web page address where the TLS encryption will be used, the company to whom the certificate was issued, the issuer of the certificate, etc. [60] The certificate comes with a public and private key, which are stored in the application’s directory. These keys are read when the process starts, and the keys are entered into the express server so that the server can use them to protect communication between the serve and the user.

During the development, the application has been running locally on the computer, that is used for development. The last stage of the project is to move the application to a Linux server that is operating within the company’s internal network, so it can be used by the employees. During the development, the application has been started with Node.js commands, but in production, a PM2 daemon process manager is utilized. PM2 can launch the application as a background process, restart it in case the Linux server restart or the application crashes, and write log information about the process. [61.]

6 Conclusion

6.1 Overview of the project

The goal of this thesis was to design and create a full-stack web application to a company, so its employees can monitor and manage customers' internet connections more easily, using a smart phone or a computer. The application has multiple features, such as creating a new connection, modifying settings of an existing connection, or executing mass changes to multiple connections at the same time. The application was supposed to be as secure as possible, easy to maintain and require minimal maintenance. Another requirement was that it can be easily updated in the future.

Node.js runtime environment was chosen as the server, as its advantages include the JavaScript programming language, versatile ecosystem, and its versatility.

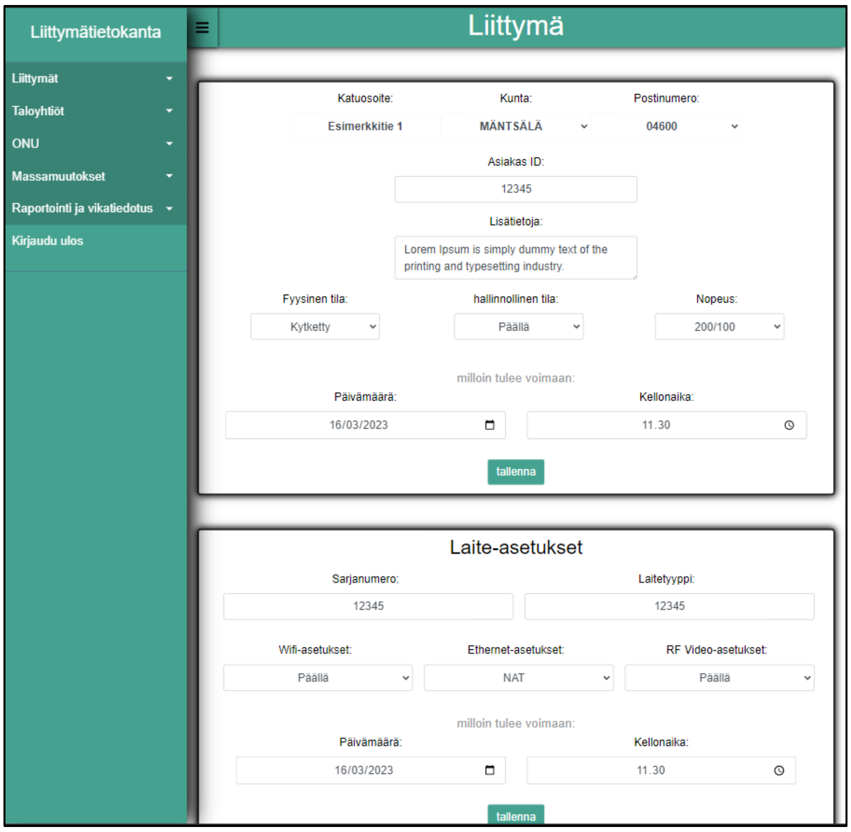
Web application's server and API was built to a Node.js process. Web application uses the API to read and modify data from the database. The database had been built on a MariaDB server, so it was necessary that the API can use this database. The MariaDB provided an npm package that was used by the API.

Both the web application and the API were built within the same Node.js process. This simplified the development and software management, as changes did not have to be made separately to multiple software programs. However, the web application uses the API instead of reading data directly from the database, even though it would have been possible to do so. This allows easy separation of the API into its own Node.js process in the future, if necessary.

Web application's frontend had to be made responsive, so it can be used on different devices, such as smart phones or computers. Bootstrap CSS framework was used for this purpose, which is designed with mobile-first approach. Bootstrap provided its own grid system, which made it easy to create a responsive frontend.

6.2 Results

The result of the project was a functional full-stack web application that met all the project requirements. The project was developed in the order outlined in the roadmap, but additional features emerged during development, such as the ability to add various additional parameters and generate report information. Despite this, the project was successfully launched on the company’s intranet, and employees were able to use the application. Figure 18 shows an example view from the page that users can use to execute changes to a connection. Most of the parameters can be altered by the users, and they are predefined and added to a dropdown list, ensuring that the selected value is exact. Users can also specify time and date when new settings will take effect. On the left-hand side is the navigation bar that users can use to navigate between different features.



Liittymä

Katuosoite: Esimerkkitie 1
Kunta: MÄNTSÄLÄ
Postinumero: 04600
Asiakas ID: 12345
Lisätietoja: Lorem Ipsum is simply dummy text of the printing and typesetting industry.
Fyysinen tila: Kytketty
hallinnollinen tila: Päällä
Nopeus: 200/100
milloin tulee voimaan:
Päivämäärä: 16/03/2023
Kellonaika: 11.30

Laite-asetukset

Sarjanumero: 12345
Laitetyyppi: 12345
Wifi-asetukset: Päällä
Ethernet-asetukset: NAT
RF Video-asetukset: Päällä
milloin tulee voimaan:
Päivämäärä: 16/03/2023
Kellonaika: 11.30

Figure 18. Screenshot of the web application

At the time of writing of this thesis, the application was not in full use, since the external software that executes changes to the physical devices, was not finished.

6.3 Improvements

After the deployment of the application, a few areas for improvement were identified. When Node.js process started, the API established a connection to the database and kept it open. However, MariaDB server terminates idling connections. This happens after eight hours [62.], so this problem was not discovered during the development. The issue arose the following day when the application suddenly stopped working.

There were two choices to fix this: Establish a new connection every time the API is used or build a mechanism that first checks the connection status and reopens it if it is closed. However, the connection should remain open throughout the entire user session, as depending on which feature the user is using in the application, the application reads multiple different pieces of information from the database. If the connection is disconnected and reopened after each read operation, it would significantly increase loading times and lower the user experience. For this reason, the second solution was chosen, where the connection status is checked and reopened if necessary.

Another area of improvement would be redesigning the fronted layout. Even though the current frontend allows users to perform necessary tasks, due to the added features in the application, “pieces” were added to the frontend, which did not result in the best possible user interface.

6.4 Future development

The application was designed to require as little maintenance as possible, while also making it easy to add possible updates. Thanks to the Node.js environment and JavaScript programming language, maintaining the application is easier since the entire project is written in one language, which lowers the threshold for other developers to make changes. Additionally, the Node.js environment’s ecosystem allows new features to be added, existing modules to be updated, or even replaced with better ones.

During the development phase, employees came up with many ideas for additional features that could be added to the application. For example, features such as checking the status of the connections, or fetching log data from external log. These would help troubleshooting situations. For these, Node.js provides an easy solution: Node.js provides child_process module, that can be used to create separate subprocesses, which can start and execute different programs and applications outside the node.js process. [16.]

The frontend could be turned into a Progressive Web App (PWA), which is a web-based application built with technologies such as HTML, CSS and

JavaScript. PWA can be installed on smartphones and computers. PWA offers a similar user experience to traditional mobile applications. When installed, PWA can among other things, save data on the user’s device and offer offline functionality. [63.] These features can shorten loading times and help operate in slow internet connections.

References

1 Chris Northwood 2018; The Full Stack developer https://link.springer.com/book/10.1007/978-1-4842-4152-3 Accessed 24 March 2023

2 Indeed; What is a web application (With benefits and jobs). https://www.indeed.com/career-advice/career-development/what-is-web-application Accessed 24 March 2023

3 MongoDB; Full Stack Development Explained https://www.mongodb.com/languages/full-stack-development Accessed 24 March 2023

4 World Wide Web Consortium; HTML & CSS https://www.w3.org/standards/webdesign/htmlcss Accessed 24 March 2023

5 Oracle; What is a Database? https://www.oracle.com/database/what-is-database/#WhatIsDBMS Accessed 24 March 2023

6 MariaDB; Temporal Tables Part 1: Introduction & Use Case Example https://mariadb.com/resources/blog/temporal-tables-part-1/ Accessed 24 March 2023

7 Red Hat 2022; What is an API? https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces Accessed 24 March 2023

8 Mozilla Developer Network 2023; Introduction to web APIs https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction Accessed 24 March 2023

9 Google; Mobile site and mobile-first indexing best practices https://developers.google.com/search/docs/crawling-indexing/mobile/mobile-sites-mobile-first-indexing Accessed 24 March 2023

10 Node.js; Introduction to Node.js https://nodejs.dev/en/learn/ Accessed 24 March 2023

11 Node.js; Overview of Blocking vs Non-Blocking https://nodejs.org/en/docs/guides/blocking-vs-non-blocking Accessed 24 March 2023

12 Mozilla Developer Network 2023; JavaScript modules https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules Accessed 24 March 2023

13 Allen Wirfs-Brock, Brendan Eich 2020; JavaScript: The first 20 Years https://dl.acm.org/doi/10.1145/3386327 Accessed 24 March 2023

14 Node.js; HTTP https://nodejs.dev/en/api/v19/http/ Accessed 24 March 2023

15 Node.js; File System https://nodejs.dev/en/api/v19/fs/ Accessed 24 March 2023

16 Node.js; Child process https://nodejs.org/api/child_process.html#child-process Accessed 24 March 2023

17 Mozilla Developer Network 2023; Node.js https://developer.mozilla.org/en-US/docs/Glossary/Node.js Accessed 24 March 2023

18 Npmjs; website https://developer.mozilla.org/en-US/docs/Glossary/Node.js Accessed 24 March 2023

19 MariaDB; ENUM https://mariadb.com/kb/en/enum/ Accessed 24 March 2023

20 Nivos website; https://www.nivos.fi/kotiin/valokuitu/valokuituyhteys-taloyhtion-asukkaalle Accessed 24 March 2023

21 Nivos website; https://www.nivos.fi/kotiin/valokuitu/mantsalan-kodit-oy-nettiliittyma Accessed 24 March 2023

22 Amazon Web Services; What Is A RESTful API? https://aws.amazon.com/what-is/restful-api/ Accessed 24 March 2023

23 MariaDB; Getting Started With the Node.js Connector https://mariadb.com/kb/en/getting-started-with-the-nodejs-connector/ Accessed 24 March 2023

24 Bootstrap website; https://getbootstrap.com/ Accessed 24 March 2023

25 Bootstrap; Cards https://getbootstrap.com/docs/5.3/components/card/ Accessed 24 March 2023

26 Bootstrap; Columns https://getbootstrap.com/docs/5.3/layout/columns/ Accessed 24 March 2023

27 Bootstrap; Breakpoints https://getbootstrap.com/docs/5.3/layout/breakpoints/ Accessed 24 March 2023

28 Cisco; What is IT Security https://www.cisco.com/c/en/us/products/security/what-is-it-security.html Accessed 24 March 2023

29 Cloudflare; Why is HTTP not secure? |HTTP vs HTTPS https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/

30 World Wide Web Consortium; The original HTTP as defined in 1991 https://www.w3.org/Protocols/HTTP/AsImplemented.html Accessed 24 March 2023

31 J. Reynolds; Assigned numbers https://www.rfc-editor.org/rfc/pdfrfc/rfc1700.txt.pdf Accessed 24 March 2023

32 Npmjs; Helmet https://www.npmjs.com/package/helmet?activeTab=readme Accessed 24 March 2023

33 Open Worldwide Application Security Project; HTTP headers Cheat Sheet https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html Accessed 24 March 2023

34 Npmjs; Passport https://www.npmjs.com/package/passport Accessed 24 March 2023

35 Passport; Strategies https://www.passportjs.org/packages/ Accessed 24 March 2023

36 Vaadata; How to Securely Store Passwords in Database? https://www.vaadata.com/blog/how-to-securely-store-passwords-in-database/ Accessed 24 March 2023

37 Npmjs; bcrypt https://www.npmjs.com/package/bcrypt Accessed 24 March 2023

38 Passport; Sessions https://www.passportjs.org/concepts/authentication/sessions/ Accessed 24 March 2023

39 Auth0; Introduction to JSON Web Token https://jwt.io/introduction Accessed 24 March 2023

40 Open Worldwide Application Security Project; SQL Injection https://owasp.org/www-community/attacks/SQL_Injection Accessed 24 March 2023

41 MariaDB; Promise API documentation https://mariadb.com/kb/en/connector-nodejs-promise-api/ Accessed 24 March 2023

42 Mozilla Developer Network 2023; Types of attacks https://developer.mozilla.org/en-US/docs/Web/Security/Types_of_attacks#cross-site_scripting_xss Accessed 24 March 2023

43 Mozilla Developer Network 2023; Content Security Policy (CSP) https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP Accessed 24 march 2023

44 Kevin Goldberg IBM 2010; Developing dynamic Web sites with CodeIgniter https://developer.ibm.com/articles/os-codeigniter/?mhsrc=ibmsearch_a&mhq=MVC Accessed 24 March 2023

45 Geeks for Geeks 2021; Benefit of using MVC https://www.geeksforgeeks.org/benefit-of-using-mvc/ Accessed 24 March 2023

46 Scythe studio 2002; Client-Side vs Server-Side rendering https://scythe-studio.com/en/blog/client-side-vs-server-side-rendering Accessed 24 March 2023

47 EJS website; https://ejs.co/ Accessed 24 March 2023

48 Microsoft; Partial Views in ASP.NET core https://learn.microsoft.com/en-us/aspnet/core/mvc/views/partial?view=aspnetcore-7.0 Accessed 24 March 2023

49 Akamai; What is a CDN (Content Delivery Network?) https://www.akamai.com/our-thinking/cdn/what-is-a-cdn Accessed 24 March 2023

50 Bootstrap; Quick start https://getbootstrap.com/docs/5.3/getting-started/introduction/#quick-start Accessed 24 March 2023

51 Mozilla Developer Network 2023; Try..catch https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch Accessed 24 March 2023

52 Node.js; Errors https://nodejs.org/api/errors.html#errors Accessed 24 March 2023

53 Mozilla Developer Network 2023; Throw https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/throw Accessed 24 March 2023

54 Visual Studio; Getting Started https://code.visualstudio.com/docs Accessed 24 March 2023

55 Visual Studio; Terminal Basic https://code.visualstudio.com/docs/terminal/basics Accessed 24 March 2023

56 Npmjs; Nodemon https://www.npmjs.com/package/nodemon Accessed 24 March 2023

57 GitHub; Get started https://docs.github.com/en/get-started/quickstart/hello-world Accessed 24 March 2023

58 Postman; What is Postman? https://www.postman.com/product/what-is-postman/ Accessed 24 March 2023

59 Npmjs; esm https://www.npmjs.com/package/esm Accessed 24 March 2023

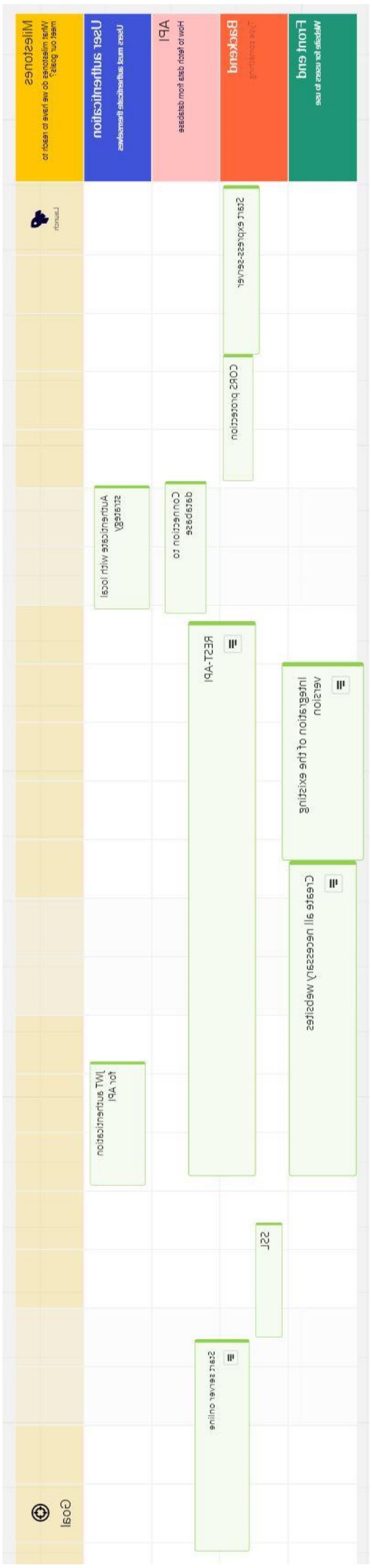
60 Cloudfare; What is an SSL certificate? https://www.cloudflare.com/learning/ssl/what-is-an-ssl-certificate/ Accessed 24 March 2023

61 PM2; Quick Start https://pm2.keymetrics.io/docs/usage/quick-start/ Accessed 24 March 2023

62 MariaDB; System Variables https://mariadb.com/docs/server/ref/mdb/system-variables/wait_timeout/ Accessed 24 March 2023

63 Google Developers 2023; What are Progressive Web Apps? https://web.dev/what-are-pwas/ Accessed 24 March 2023

Roadmap


Milestones
What milestones do we have to reach to meet our goals?
User authentication
Users must authenticate themselves
API
How to fetch data from database
Backend
Front end
Website for users to use
Launch
Start express-server
CORS protection
Authenticate with local strategy
Connection to database
REST-API
Integration of the existing version
Create all necessary websites
JWT authentication for API
SSL
Start server online
Goal