

Md Alamin

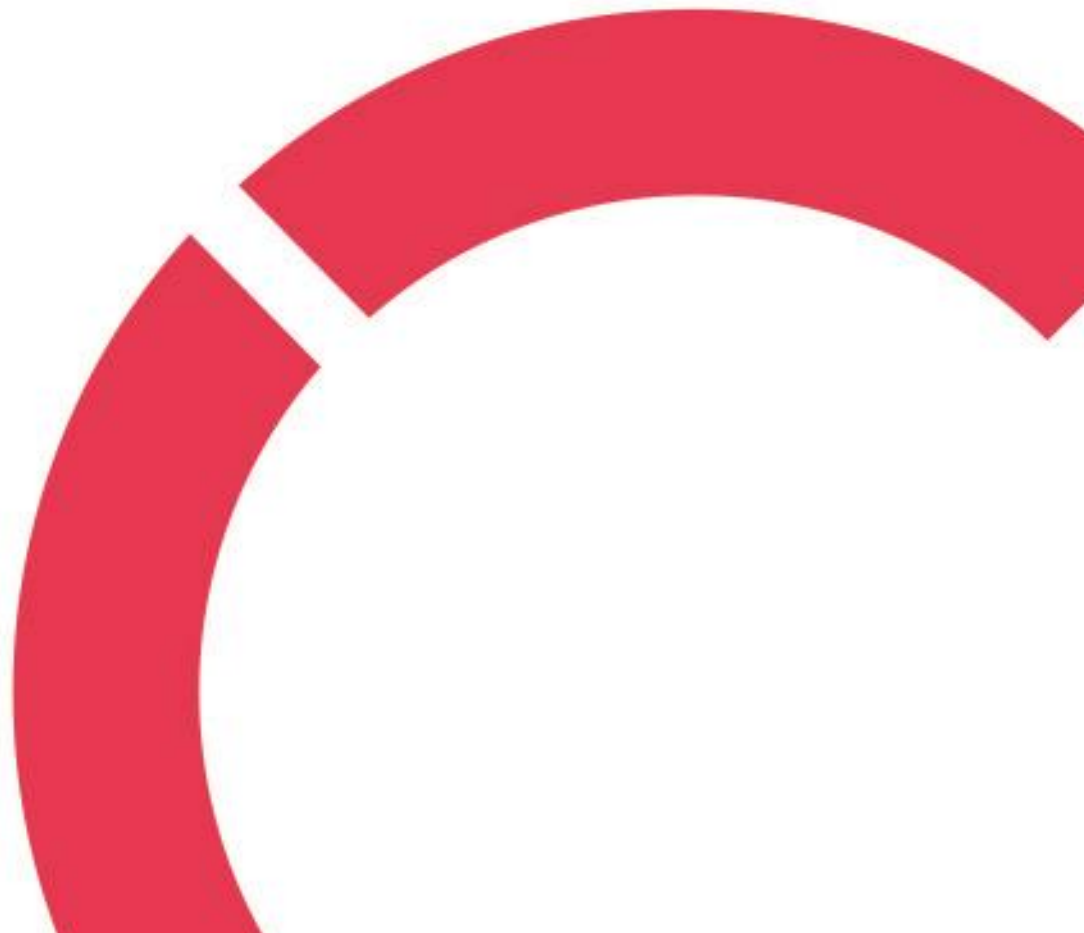
**A SOCIAL PLATFORM FOR SOFTWARE DEVELOPERS: USING
MODERN WEB STACK MERN**

Thesis

CENTRIA UNIVERSITY OF APPLIED SCIENCES

Information Technology

December 2022



ABSTRACT

Centria University of Applied Sciences	Date December 2022	Author Md Alamin
Degree programme Information Technology		
Name of thesis A SOCIAL PLATFORM FOR SOFTWARE DEVELOPERS: USING MODERN WEB STACK MERN		
Centria supervisor Kauko Kolehmainen	Pages 42	
Instructor representing commissioning institution or company Kauko Kolehmainen		
<p>The aim of the thesis was to explore the MERN stack and JavaScript library Redux to build a social platform application for developers. The thesis has been shown how it works for frontend and backend. Why a developer needs to know the MERN stack and what made it so popular. or organizations, there are several reasons why the MERN stack is a good choice. From the ease and speed at which it helps developers to create and maintain applications to its reliability to the excellent usability of developers, and interactive elements for web pages, enhancing the user experience.</p> <p>The thesis is divided into two parts, theoretical and practical. The theoretical part will cover the MERN stack. The MERN stack is short for ReactJS, Express JS, Node JS, and MongoDB. And the practical part will cover details the project idea, flowchart, and implementation details.</p> <p>After months of doing research, the project was built successfully and fully works functionally. The project created a social media for software developers platform where around the world developers can interact with each other. For example, they can share their knowledge as a post, like, and comment. Also, they can build their professional profiles. The application was created from scratch. One of the interesting points about the application is that the application will be fully real-time (SPA) with no loading time, data will render in a few seconds.</p>		

Key words

Express JS, MongoDB, NodeJS, REST API, React JS, Redux JS, Social network.

CONCEPT DEFINITIONS

MERN	MongoDB, Express, React, Node
API	Application Programming Interface
URL	Uniform Resource Locator
NPM	Node Package Module
UI	User Interface
HTML	HyperText Markup Language
JS	JavaScript
CSS	Cascading Style Sheets
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
JSX	JavaScript XML
AJAX	Asynchronous JavaScript And XML
XML	Extensible Markup Language

ABSTRACT
CONCEPT DEFINITIONS
CONTENTS

1 INTRODUCTION	1
2 FRONTEND DEFINITIONS	2
2.1 Hypertext Markup Language (HTML)	2
2.2 Cascading Style Sheets (CSS).....	2
2.3 JavaScript	3
2.4 React	3
2.4.1 JSX	4
2.4.2 Virtual Dom	4
2.4.3 Components	5
3 BACK-END DEFINITIONS	6
3.1 Node Js	6
3.1.1 Node Package Manager (NPM)	6
3.2 Express	7
3.2.1 Asynchronous Programming	7
3.2.2 JWT.....	8
3.2.3 Understanding middleware	8
4 DATABASE	9
4.1 MongoDB	9
4.2 Mongoose	10
5 IMPLEMENTATION OF THE PROJECT	11
5.1 Tools and Technologies	11
5.2 Backend Creation	11
5.2.1 Server Creation	12
5.2.2 Database Creation	15
5.2.3 Authentication and Authorization	18
5.2.4 Controllers	19
5.2.5 Routes.....	23
5.3 Frontend Creation	25
6 CONCLUSION	34
REFERENCES	36

FIGURES

Figure 1. Workflow of backend	12
Figure 2. NPM package JSON.....	13
Figure 3. declare package and library.....	14

Figure 4. Database connection and Main API routes	14
Figure 5. Server PORT and running server	15
Figure 6. User Schema.....	16
Figure 7. Post schema.....	17
Figure 8. MongoDB Objects.....	17
Figure 9. Authentication.....	18
Figure 10. Controller workflow.....	19
Figure 11. User registration.....	19
Figure 12. Users check.	20
Figure 13. Encrypt password and JWT token activation.....	20
Figure 14. Sending email method.....	21
Figure 15. Email sending and Verification.....	21
Figure 16. Routes testing from the postman	22
Figure 17. followers and following connections in user object.....	22
Figure 18. Declare library and modules.....	23
Figure 19. User routes.....	24
Figure 20. MyProfile route data.....	24
Figure 21. Post routes.....	25
Figure 22. Register form and API connection.....	26
Figure 23. Frontend visualization.....	27
Figure 24. Signup.....	27
Figure 25. Sent activation email and activation page.....	28
Figure 26. Login page and Dashboard.....	29
Figure 27. Create profile page.....	30
Figure 28. User education.....	31
Figure 29. Experience page.	32
Figure 30. Dashboard.	33

1 INTRODUCTION

The thesis consists primarily of two major parts. The theoretical part focuses on giving a deep understanding of the MERN technologies stack and one of the JavaScript library Redux JS. The MERN stack, short for M, stands for MongoDB, E for ExpressJS, R for ReactJS, and N for NodeJS. Express is a server-side web framework, and NodeJS is a popular and powerful JavaScript server platform. And ReactJS contributed to the client side.

The thesis showed how users could register, log in, and store their information in the database. Each component in the stack and the associated technologies. The MERN stack makes it very scalable, so many users can interact with this app simultaneously. This web application ("Developer Meet") is built to connect developers worldwide, share their experiences, and gather others' experiences.

2 FRONTEND DEFINITIONS

Every software has a frontend and a backend component. The frontend, also known as the client side, refers to the user interface that the user interacts with. In the past, the frontend was simply responsible for the visual aspects of a website, such as its layout, color scheme, and font choices. However, with the evolution of programming languages and tools like React, Vue, and Angular JS, the frontend has become more dynamic and capable of handling more complex tasks. One of the primary responsibilities of the frontend is to retrieve data from the backend through an application programming interface (API). The frontend then presents this data to the user in a visually appealing and intuitive way. For example, on a social media platform like Facebook, the frontend might display a news feed component that retrieves updates from the backend and presents them to the user in real time. In summary, the frontend plays a crucial role in how users interact with and navigate a website or application. It combines design skills and programming knowledge to create a dynamic and user-friendly experience. (Le 2020.)

2.1 Hypertext Markup Language (HTML)

HTML is a HyperText Markup Language. HTML uses tags to identify a web page's elements and tells a browser how to lay a web page out on a screen. It uses tags to define elements of a webpage. Without HTML, nothing is possible in the software, especially the consumer base. So, HTML is a most popular language and what people see in the user interface. For example, people can post, like, comment and retweet in Twitter. It happens because of HyperText Markup. HTML is a design of the text. Nowadays, people can use HTML in many ways such as- one JavaScript library called react comes with the JSX. HTML and JSX are quite different, but they work more similar. JSX (JavaScript Extension) is a React extension that allows writing of JavaScript code that looks very similar to HTML and developers can use HTML in PUG. (Larsen 2013.)

2.2 Cascading Style Sheets (CSS)

CSS is a Cascading Style Sheet. It describes how HTML objects are displayed on the browser. CSS is the document style. For example, page layouts, colours, and text fonts are all determined with CSS. Such as on Twitter, Facebook, or Instagram, user can see responsive and mobile views, lots of colours and text styles. If a developer needs some colours, buttons, text, menu or any style, developer can

make these by CSS. Nowadays, CSS is quite different from one of the libraries that came with Tealium CSS. Tealium CSS is like SASS. So basically, SASS combines with the class. If developer write pure CSS, it takes long time than SASS. Developers try to ignore customs CSS nowadays. They use Tealium CSS, and Tealium CSS is something that if developers write one or two-line codes, they can see major changes in the front end. (Larsen 2013.)

2.3 JavaScript

JavaScript is one of the most popular language. Before 2014, JavaScript was a complicated programming language. Back in 2014, JavaScript developed with the new structure ES5 and ES6. So, JavaScript widely use around the world and 90% of the applications is now using JavaScript. Without JavaScript, consumers or public based software are impossible to develop. So, JavaScript language play vital role in the framework and library. JavaScript is like the parent of many libraries or frameworks. And JavaScript appears with asynchronous systematically. Back in 2014-15, Facebook was quite slow, and nowadays, Facebook, Twitter, and YouTube are faster because of the ES6's asynchronous which is updated version of JavaScript. When a user clicks the Like button on a post, at the mean time another user also clicks on that. Then asynchronous don't wait for others. If your internet is faster than another user, your Like will be added first. Before it would work as- if a user would click on the Like button and JavaScript kept the user Like on proceed, then show serial by serial. Nowadays, in modern JavaScript, there is no waiting time and it works faster. On the other hand, JavaScript is a language with an extensive community in Google. If we compare C++, C, C# Python and JavaScript, among them Python and JavaScript, is quite popular. Whenever developers face some critical or challenging time, they can easily find out solutions from Google. Nowadays, JavaScript is more modern than past, a developer can write one function three times or four times, but JavaScript is reusable today. (Shute 2019.)

2.4 React

React.js is one of the most popular and open-source JavaScript libraries. React is a flexible, reliable, and declarative library for building user interfaces specifically for single-page applications. And React allows us to create reusable UI components. It is a component-based front-end library which creates large web applications that can change data without reloading the page. The main purpose of React is to be customizable, easy, and fast. React was created by Jordan Walke, a software engineer working for Facebook. React was first deployed on the Facebook newsfeed and later used in its products like

Whatsapp, Instagram, Uber, Airbnb, Pinterest, Netflix, Amazon, Twitter, and many other companies. Facebook developed React in 2011 but it was released to the public in 2013. For instance, the number of views of a particular video, in a platform like Youtube, can be seen automatically, without having to wait for it to load. Furthermore, without having to reload the page, the number of reactions, on a certain post or video, can be obtained. This phenomenon is resulted by the React Js. Facebook, Instagram, and Twitter as a whole, with their trillions of posts and billions of users are counting quadrillions of likes, comments, and shares every day, which necessitates numerous speeds. This speed can be achieved with React JS. (Cory 2015.)

2.4.1 JSX

JSX allows us to write JavaScript Html code. JSX converts HTML tags to react elements. JavaScript extension XML and JSX syntax use in React Js. JSX is an extension to write XML code for elements and components. JSX tags have a tag name, attributes, and children. JSX is not a necessity to write react applications. Developers can use React without JSX. And JSX makes react code simpler and more elegant. JSX ultimately transpiles points to an understandable pure JavaScript browser. JSX (JavaScript Extension) is a React extension that allows the writing of JavaScript code that looks very similar to HTML. Which makes it easier to create templates. (Richey 2019.)

2.4.2 Virtual Dom

The Virtual DOM is a memory of the real DOM that acts as an intermediary between the state of the application and the DOM of the graphical interface that the user sees in the interface. The document's structure is defined by it, and the web page or document is converted by the browser. Which is the object representation of the document and can be changed or manipulated using a scripting language like JavaScript. React makes two copies of Virtual Dom from HTML DOM. The virtual Dom is the main reason for the excessive speed of the React. (Cory 2015.)

2.4.3 Components

Component is considered core building block of any React application, and a single app usually consists of multiple components. Interestingly, React JS allows to use custom React components inside another concept. Everything in React is a component that makes building UIs much easier. Mainly React has two types of components one is Functional component and the other one is Class component. Because they do not have memory, functional components always receive data from other components. Additionally, Class Components possess memory and can store data. (Richey 2019.)

3 BACK-END DEFINITIONS

Backend refers to everything data related. The controller is the location where logical operations occur. The backend takes responsibility for security and what kind of data and logic goes to the front-end. The backend provides some API. And API is a form of integration. The frontend does not comprehend everything in the backend, it just comprehends what the backend says. They comprehend one another because of the API. The backend is the area that houses the business logic, handles security concerns, and maintains a connection to the database. (Vickler 2021.)

3.1 Node Js

Node JS is a JavaScript runtime environment built on Chrome's V8 JavaScript engine that allows to run JavaScript code on the server side. NodeJS is open source and open community. Any company or individual does not own it. It works on all major operating systems, including Windows, macOS, and Linux. The fact is the OS of choice for many embedded applications. NodeJS gives us access to databases, storage systems, authentication services (such as Auth) and many other capabilities. Node developers can find many good resources on the web and learn more about Node.js. One good starting place is the Node.js website. Another good place to learn more about Node.js is in a blog post series. NodeJS also has many libraries. These libraries can help you with tasks such as connecting to a database. (Herron 2016.)

3.1.1 Node Package Manager (NPM)

Node Package Manager is the default package manager for JavaScript runtime environment Node.js. It is completely written in JavaScript. We can install any package using NPM on a Node.js project. A package contains all the source code of modules and can be included in a Node project based on requirements. NPM has two types of packages, public package, and private package. Public package is like, a developer can create and publish packages that anyone can download and use in their own projects. Private package is a developer can publish a package the is only visible to you and chosen collaborator. Private packages always have a scope, and scoped packages are private by default. NPM

basic commands `npm i <packageName>` used to install a package from json file in local environment. And `npm i -g <packageName>` used to install a package from json file in global environment. The command `npm un <packageName>` used to uninstall a package in local environment. Some Node.js frameworks Express.js, Koa.js, Socket.io, Nest.js. (Mardan 2014.)

3.2 Express

Express.js is one of the most popular backend web application NodeJS framework. It is very easy to use and provides a wide range of features that make it perfect for web and mobile application development. Express.js has an extensive web community, which is used for commercial applications. The confidence to use this framework for any project, no matter how big or small, is given to developers by this framework. With Express.js, you can take advantage of various support packages and additional features. This will help developers create better programs. However, it does not slow down the performance of NodeJS. Express.js is a core component of one of the most popular platforms today that is NodeJS. Express.js was introduced by TJ Holowaychuk and was first released in 2010. Error handling, HTTP requests, sessions, and routing were all managed by Express JS. In addition, it reduces the amount of time for programmers and helps in delivering apps more quickly and efficiently. Using Express, developers can easily connect to databases like MySQL, MongoDB, and Redis. It is very easy to learn and manipulate Express.js is for JavaScript developers. (Brown 2019.)

3.2.1 Asynchronous Programming

Asynchronous programming is a programming method that, has been used to speed up our code and make it more responsive. Asynchronous helps us to move on to another task without waiting for the first task to finish. Asynchronous programming is widely used in application and web programming, as it allows an application to continue running without blocking input/output operations. In synchronous programming, the program flow is blocked until an operation is completed. In asynchronous programming, the program flow is not blocked; instead, the program registers a callback function to be called when the operation is completed. Asynchronous programming can be implemented in various ways, such as using threads, callbacks, or promises. The native implementation of promises was added to JavaScript in 2015. It is a solution to the problems inherent in callback-based asynchronous programming. A Promise is an object that represents a task that will be completed at some point in the future.

Promises have two key methods `then()` and `catch()`. `Then()` method is used to specify what should happen when the task represented by the promise is completed. And `catch()` method is used to specify what should happen if the task represented by the Promise fails. `Async/Await` is a syntax extension built on top of promises. It makes working with asynchronous tasks in JavaScript much easier and more readable. The `async` keyword is used to create a new `async` function. (Wilson 2018.)

3.2.2 JWT

JSON Web Tokens (JWTs) define a way to securely transmit the information as a JSON object in a compact and self-contained way. self-contained means for sending JSON data to the database and returning JSON object data. Additionally, the sending data is protected by digitally signing it with the HMAC method or by signing it with RSA using a public/private key pair. The digital signature on this information guarantees its authenticity and trustworthiness. In addition to providing secrecy, JWTs can also be encrypted. Payloads are encrypted so anyone cannot see the token. There are three main parts of a JWT header, payload, and signature. Parts are divided with a dot. The header contains the signing algorithm (HMAC/RSA) and the token type (JWT). Payloads contain information about entities in the form of claims. Claims are separated into three types: public, private, and registered. (Brown 2019.)

3.2.3 Understanding middleware

Middleware can be used to verify a user's credentials, to verify that a user is authorized to access a resource, to log requests, or to perform any other action that needs to be performed before the request is handled by the controller. In many cases, the controller will use a variety of middleware to verify the user's credentials and to verify that the user is authorized to access a resource. Some middleware options that used include, session id, cookies, URL, and authorization. A session ID is used to verify the user's credentials and to ensure that the user is authorized to access the resource. Cookies are used to store user data and to ensure that the user is authorized to access the resource. A URL is used to verify the user's credentials and to ensure that the user is authorized to access the resource. And the authorization is used to ensure that the user is authorized to access the resource. (Wilson 2018.)

4 DATABASE

A database is a place where data is stored. Consequently, user can easily access and modify the data from the database. A system that contains database is called a database management system (DBMS). Databases are very essential components for all modern applications. As users, we interact with many databases daily by visiting websites and applications on our phones. Phone numbers, other contact information, and numerous other details that are stored in a database by the mobile devices of people are an example of a database. There are a few types of databases that are very important and popular like, NoSQL Database, Relational Database, Cloud Database, Graph Database, Distributed Database. (Stephens 2008.)

4.1 MongoDB

MongoDB is a document-oriented and NoSQL database. The principal explanations behind picking MongoDB are they have an enormous informational index and MongoDB is truly versatile. Document-oriented databases are powerful because they allow applications to evolve quickly. Assuming an application records client orders, the application may begin with a straightforward data structure that stores the name of the customer, quantity, and product order. The application might add new fields to the order document, such as the customer's address, order date, and shipping method. The application can also add new data types to the customer document, such as the customer's credit rating or preferences. MongoDB is a NoSQL database, and the NoSQL database is a non-relational database. A NoSQL database does not use tables and does not require a fixed schema. A NoSQL database is easy to scale. There are several reasons why MongoDB is the most widely used NoSQL database. The first reason is that MongoDB is a document-oriented database. This means that it stores data in documents, which are like rows in a table. Each document has a unique ID and has number of fields. This makes it very flexible, as we can add or remove fields as needed. The second reason is that MongoDB is very scalable. It can be easily scaled up or down, depending on our needs. The third reason is that MongoDB is very fast. It can handle large amounts of data very quickly and can be used for real-time applications. (Vohra 2015.)

4.2 Mongoose

NodeJS and MongoDB communicate with each other through the Mongoose. Mongoose validates schemas and helps to connect code objects with MongoDB objects. Mongoose provides Object Data Modeling (ODM) to NodeJS. It helps to connect code objects with MongoDB objects. Mongoose also validates schemas. MongoDB acts as a middleman between objects in code and objects in MongoDB, ensuring that data in the database is valid. Mongoose contains methods and functions to help NodeJS and MongoDB understand each other better. (Vohra 2015.)

5 IMPLEMENTATION OF THE PROJECT

This application has a signup system so that users can create new accounts and join the Developers Meet platform allows developers to interact with each other around the world. Additionally, they can make their professional profiles and learn by commenting and posting on other posts of the developers. The application implementation consists of two separate parts, the backend, and the front end. This thesis covers a limited area. A fascinating aspect concerning the application is, it is planned scalably, API, tools, and utils have also been created. All the things were created individually. It has validation, authentication, middleware, controller, scheme, micro-service, and email confirmation. It is from scratch and fully real-time (SPA) with no loading time. In a few seconds, the data will be rendered.

5.1 Tools and Technologies

The project is built using MERN stack and Redux JS. Mostly in the backend, technology(stack) used node Js, Express JS, and MongoDB. This application utilized React, Redux, HTML, CSS, and JavaScript on the frontend. In this application, the user's data is stored in MongoDB. The main reason for choosing MongoDB it has large data set. MongoDB is the most popular NoSQL database and is very scalable. Frontend API is operated by Redux JS. MERN stack has gained popularity nowadays. More applications are developed using the MERN stack. In terms of tools, we require a source code editor, Visual Studio Code (VSCode), Google Chrome, Postman, and MongoDB. Postman is used to performing API calls, while MongoDB shows us what kind of data have in our database.

5.2 Backend Creation

The backend of a software application is responsible for managing and processing data, as well as handling server-side tasks and functionality. One common way to build the backend of a web application is by using a JavaScript runtime environment called Node.js, along with a web application framework called Express.js. Node.js allows developers to write server-side code in JavaScript, which can be executed on the server rather than in the user's browser. Express.js is a lightweight framework built on top of Node.js that simplifies the process of building and deploying web applications. It provides a set of tools and features for routing, middleware, and other common backend tasks. To store data, the backend can use a database management system like MongoDB. This NoSQL database stores data in a flexible,

JSON-like format and can scale easily to handle large amounts of data. The Mongoose library is often used to connect to and interact with a MongoDB database from a Node.js application. Finally, the backend may expose certain endpoints or routes that the frontend (client-side) can access through an API. These endpoints allow the frontend to send and receive data from the backend, enabling users to interact with the application and access its functionality.

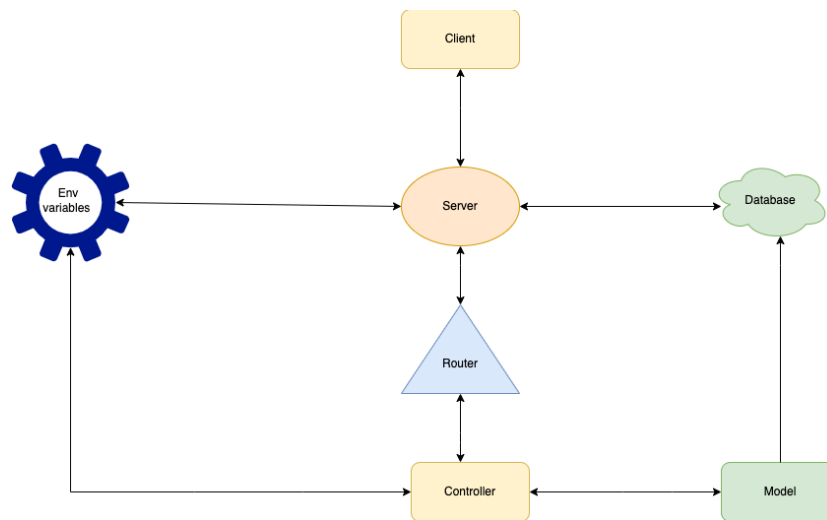


Figure 1. Workflow of backend.

Figure 1 shows the basic server architecture and how the server works. whenever a user clicks on a view, such as a dashboard, personal profile, edit, comment, or share. it is hit the API, and the API connect with the server. The server has a route connection. The router has a connection with the controller. The controller is connected to the router. Additionally, the controller contains all necessary user-specific logic. As a result, the controller checked the environment variables and matched all the model's data. All security is checked by the environment, and the model connects to the database. Finally, the user receives the data via API from the environment and database.

5.2.1 Server Creation

The server side, which also refers to the application's internal workings, is typically referred to as the backend. Although it is not directly involved with the user, this section of the website oversees manipulating and storing application data. An explanation of the technologies used to create this application is included in this section. The server creation requires the installation of nodeJs. And then, compile the command `npm init` for building the nodeJs project. This command will add basic information about the project in packageJson. (Figure 2.)

```

() package.json > {} dependencies
 1  {
 2    "name": "developer-meet",
 3    "version": "1.0.0",
 4    "description": "",
 5    "main": "index.js",
 6    > Debug
 7    "scripts": {
 8      "test": "echo \"Error: no test specified\" && exit 1",
 9      "start": "node server.js",
10     "server": "nodemon server.js"
11   },
12   "repository": {
13     "type": "git",
14     "url": "git+https://github.com/mdalamin-eu/developer-meet.git"

```

Figure 2. NPM package JSON.

Afterwards, install Express JS and create an entry file called server.js, and declare `express`, `mongoose`, `body-parser`, `cors`, `morgan`. `Mongoose` translates between objects in code and validates schema, and it represented of those objects in MongoDB to Node JS. `Body-parser` is a middleware service, and it is parsing the incoming request bodies in a middleware before handling. `Cors` is used to secure web APIs, and it allows any domains to make requests against web API. With `Morgan` developer can debug and create log files for HTTP requests and errors in Node JS and Express JS. `Dotenv` allows the creation of secret keys for source code and keeps them from the public. (Figure 3).

```

JS server.js > ...
 1  const express = require('express')
 2  const mongoose = require('mongoose')
 3  const bodyParser = require('body-parser')
 4  const app = express()
 5  var cors = require('cors');
 6  const morgan=require('morgan')
 7  require('dotenv').config()

```

Figure 3. declare package and library.

Figure 4 shows the connection between the server and the database. During this part, Mongoose is used since it contains the necessary methods to work with MongoDB. We can use Mongo driver to interact with MongoDB. It is more convenient with Mongoose. As can be seen in Figure 4 main API routes. They are later divided into sub-routes inside the routes folder. In this part, when a user clicks on a route, it takes them to the routes of the router folders. For example, if a user hits `/api/users` route or `schools`, `posts` routes, it hits all the sub-routes located inside the `routes/user.js` file, `post.js` file, and `schools.js` files.

```

15  mongoose.connect(process.env.DATABASE_CLOUD, { useNewUrlParser: true })
16  .then(() => console.log('DB connected'))
17  .catch((err) => console.log(err))
18
19  // API
20  app.get("/", (req, res) => res.send("API running"))
21
22  app.use("/api/users", require("./routes/user"));
23  app.use("/api/posts", require("./routes/posts"));
24  app.use("/api/schools", require("./routes/schools"))

```

Figure 4. Database connection and Main API routes.

Figure 5 shows the app port and successfully run message. This code sets up a server to listen for incoming connections on a specified port. Here define which port this project will run `const PORT = process.env.PORT || 8080`. This line defines a constant called `PORT` and sets it to the value of the `PORT` environment variable if it exists. If the `PORT` environment variable is not set, it defaults to 8080. The `PORT` environment variable is often used to specify the port number on which a server should listen for incoming connections. This allows to easily change the port number without modifying the code. The second line code, `app.listen(PORT, () => (console.log(App listening on ${PORT})))`, This line tells the server to start listening for incoming connections on the port specified by the `PORT` constant. The second argument is a callback function that will be executed when the server starts listening for connections. In this case, the callback function simply logs a message to the console indicating that the server is listening on the specified port. The result of running the project and output looks like this (Figure 5).

```

26 | const PORT = process.env.PORT || 8080
27 | app.listen(PORT, () => (console.log(`App listening on ${PORT}`)))

```



```

[nodemon] 2.0.16
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
morgan deprecated undefined format: specify a format server.js:10:9
morgan deprecated default format: use combined format server.js:10:9
express-validator: requires to express-validator/check are deprecated. You should just use require("express-validator") instead.
App listening on 8080
(node:3704) DeprecationWarning: current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discovery and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:3704) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
DB connected

```

Figure 5. Server PORT and running server

5.2.2 Database Creation

For connecting MongoDB to Express Js, it is necessary to install a Mongoose package. Mongoose is used to create a schema for MongoDB. Schema interfaces are used to define models. The schema allows the users fields stored to define values and validation methods. In this application, the schema will store all user information. There are many different types and properties for each field in this schema. Name, email, and password are the three mandatory fields, and all fields are strings. In schools, following, followers fields have used square brackets. Square brackets use for tables, column names, or identifiers. So here one user can study more schools, and users can follow many people and he may have many followers. Additionally, the user can post multiple times. (Figure 6).

```

const mongoose = require('mongoose')
const Schema = mongoose.Schema;
const UserSchema = new mongoose.Schema({
  name:{
    type:String,
    required:true
  },
  email:{
    type:String,
    required:true,
    unique:true
  },
  password:{
    type:String,
    required:true
  },
  birthdate:{
    type:Date
  },
  avatar:{
    type:String
  },
  phone:{
    type:Number
    //required:true
  },
  following: {
    user: {
      type: Schema.ObjectId,
      ref: 'User'
    }
  },
  followers: [
    {
      user: {
        type: Schema.ObjectId,
        ref: 'User'
      }
    }
  ],
  posts: [
    {
      type: Schema.Types.ObjectId,
      ref: "Post"
    }
  ],
  schools:[
    {
      type: Schema.Types.ObjectId,
      ref: "School"
    }
  ],
  date:{
    type:Date,
    default:Date.now
  }
});
module.exports = User = mongoose.model("U

```

Figure 6. User Schema.

In this DB schema has been used user post details. Each field in this schema has many different types and properties such as post, post id, postdate, post likes and comments objects. When a user comments on a post it automatically added user details, name, text, avatar, and date. As we know, MongoDB uses objects instead of structured tables. These objects also used connections, references, and data types of properties (Figure 7).

```

user: {
  type: Schema.Types.ObjectId,
  ref: "User"
},
likes: [
  {
    user: {
      type: Schema.Types.ObjectId,
      ref: "User"
    }
  }
],
comments: [
  {
    user: {
      type: Schema.Types.ObjectId,
      ref: "User"
    },
    text: {
      type: String,
      required: true
    },
    name: {
      type: String
    },
    avatar: {
      type: String
    },
    date: {
      type: Date,
      default: Date.now
    }
  }
];

```

Figure 7. Post schema.

MongoDB database objects are depicted in Figure 8. In the database properties, we can see the user object id, name, email, phone number, hash password, and avatar, as well as linked posts, schools, following, and followers. When a user enters a plain password, a hashed or encrypted version of the password is saved in the database. A user can have many posts, schools, following and followers that are all linked together, because of all data are string. (Figure 8)

```

▶
  _id: ObjectId('634c4e8c2e939d2a4ae27fde')
  > posts: Array
  > schools: Array
    name: "Alamin"
    email: "mdalamin.ect@gmail.com"
    phone: 1834313691
    password: "$2a$10$kxFc1p.2s6IWrfD.Y00U/u3CrohV01mXhFduPIV0sWl29uIkBwPmW"
    avatar: "http://www.gravatar.com/avatar/f4f8746e883c4fale86c2baedb952a7a?s=200&r=pg&..."
  > following: Array
  > followers: Array
    > 0: Object
      _id: ObjectId('63558b850e3e0e2db1fe2f9d')
      user: ObjectId('634c42082e939d2a4ae27fd9')
      date: 2022-10-16T18:33:48.794+00:00
      __v: 1

```

Figure 8. MongoDB Objects.

5.2.3 Authentication and Authorization

Authentication is the process of identifying users or anyone else and getting access to logging requests or performing any other action that needs to be completed before the authentication handles the request. It is the first step of a system with a user element. After identifying the user's identity, the system redirects to the user's profile. Hence, Authorization gives the permissions to the current account that has just been authenticated.

```
middleware > JS authguard.js > ...
 1  const jwt = require('jsonwebtoken')
 2  |
 3  module.exports = (req, res, next) => {
 4  |   const token = req.header('x-auth-token');
 5  |   if(!token) {
 6  |     return res.status(401).send({
 7  |       msg:"No token, authorization denied"
 8  |     })
 9  |   }
10  |   // Verify token
11  |   try {
12  |     const decoded = jwt.verify(token, process.env.JWT_SECRET);
13  |     req.currentuser = decoded;
14  |     next()
15  |   } catch (err) {
16  |     res.status(401).json({ msg: "Token is not valid" });
17  |   }
18  | };
```

Figure 9. Authentication

In the authentication method, we can see here is checked `x-auth-token` key in request object. If there is no `x-auth-token` provided, then the function throws an error that “No token, authorization denied”. And if a token is found, then it verifies with the JWT secret key, which is stored in the JWT token. After verifying, `x-auth-token` add a new key `current user` with the value of that verification in the request object and proceeded to the next function. As request object is wrapped inside the try-catch block, if any error occurs, the function will return an error saying, "Token is not valid." (Figure 9).

5.2.4 Controllers

Controllers are responsible for handling incoming requests and returning responses to the client. A controller's purpose is to receive specific requests for the application. Controllers receive the request from the user, and then the controller function gets the requested data from the models and return it to the user to view in the browser (Figure 10).

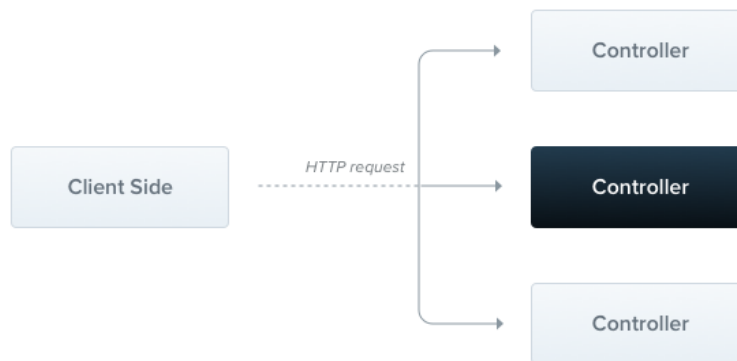


Figure 10. Controller workflow.

In this figure, controller has been used for user registration. Firstly, async function checks for any possible errors in the request and if any error is found, then sends error status 400 to the frontend. Otherwise, it passes the request and response to the next functions (Figure 11).

```

exports.registerAdd=
async (req, res) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }
}
  
```

Figure 11. User registration.

When a user registers the signup form and if this user's given email is found in the database, it sends an error message saying, "User already exists". Otherwise, the function creates a new user with the given information name, email, password, avatar, phone, birthdate, with the avatar size and user can add the gravatar URL. (Figure 12).


```

25     const { name, email, password, phone, birthdate } = req.body;
26
27     try {
28       let user = await User.findOne({ email });
29
30       if (user) {
31         return res
32           .status(400)
33           .json({ errors: [{ msg: "User already exist" }] });
34       } else {
35         const avatar = gravatar.url(email, {
36           s: "200",
37           r: "pg",
38           d: "mm"
39         });
40         user = new User({
41           name,
42           email,
43           password,
44           avatar,
45           phone,
46           birthdate
47         });

```

Figure 12. Users check.

As shown in figure (Figure 13), a plain password has been hashed by the `bcrypt` method with a salt which is generated with `bcrypt.genSalt` with a specific length value. As part of the `authenticate` method, the function receives a plain password and hashes it, afterwards compares it with the encrypted version stored in the database. And creates a payload for signing a JWT token with newly created user details with `JWT_ACCOUNT_ACTIVATION` secret and adding an expiry date of 5 hours.

```

49     const salt = await bcrypt.genSalt(10)
50     user.password = await bcrypt.hash(password, salt)
51
52     const payload = { id: user.id, name: user.name, avatar: user.avatar, password: user.password, phone: user.phone, email: user.email, birthdate: user.birthdate };
53
54
55     const token = jwt.sign(payload, process.env.JWT_ACCOUNT_ACTIVATION, {
56       expiresIn: '5h'
57     });

```

Figure 13. Encrypt password and JWT token activation.

The registerUserEmail function generates mail object using AWS mail service ses to send verification email to user mail. After sending the email, the method checks whether it was delivered or not. If it has been sent to the user then the method shows a message saying, “Email has been sent to user email, Follow the instructions to complete your registration”. Otherwise, the error message “Sorry, We Could Not Verify Your Email” will be displayed if this method finds any errors. (Figure 14).

```

//send email
9  const params = registerUserEmail(token, email) |
0  const sendEmailOnRegister = ses.sendEmail(params).promise();
1  sendEmailOnRegister.then(data => {
2
3    res.json({
4      message: `Email has been sent to ${email}, Follow the instructions to complete your registration`
5    })
6
7  }).catch(error =>{
8    console.log("Ses email on register", error);
9    res.json({
0      message: 'sorry, We could not verify your email'
1    })
2
3  })
4
5  }

```

Figure 14. Sending email method

The sendEmailOnRegister method (Figure14) has successfully sent a verification email via AWS email service ses to the user email mdalamin.eu@gmail.com for registering user account in the DeveloperMeet. Afterwards, the user Alamin received the verification email and if the user clicks the link. User account will be verified and returned the message, “You are registered”. If user exist on the database, then the message saying, “User already registered” (Figure 15).



Figure 15. Email sending and Verification.

After successful authentication by email, all data of the user is transmitted to the database. In figure 16 the login routes have been tested, and sending email, password body to `/api/users/login`. After that, the user was able to successfully log in, and send user id, name, email, token, and loggedin status true (Figure 16).



```

POST localhost:8080/api/users/login

1- {
2-   "email": "mdalamin.ect@gmail.com",
3-   "password": "Alamin123"
4- }

Body    Cookies    Headers (9)    Test Results

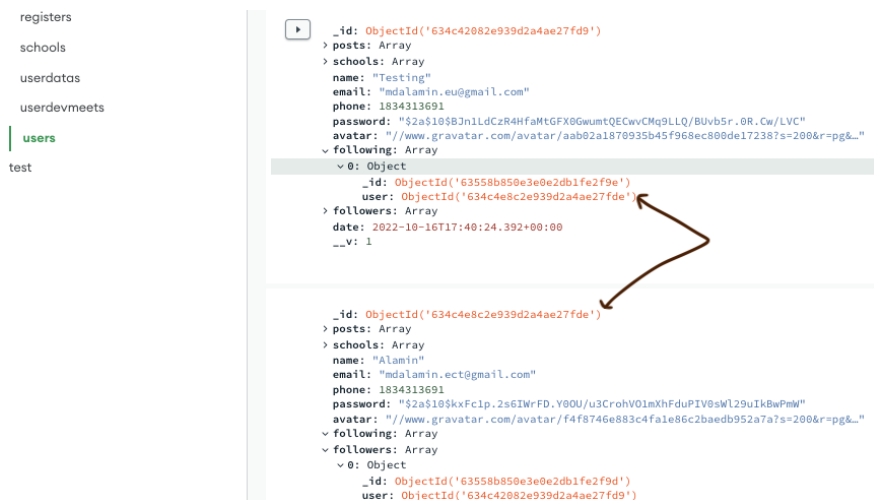
Pretty  Raw    Preview    JSON

1- {
2-   "user": {
3-     "id": "634c4e8c2e939d2a4ae27fde",
4-     "name": "Alamin",
5-     "email": "mdalamin.ect@gmail.com",
6-     "loggedin": true
7-   },
8-   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjZyNGM0ZThjMmUzMzlkMmE0YUlyYnZkZSIsIm5hbWUiOiJlJmVjdB8nbWVpbC5jb20iLCJsb2N0IjleHAiOiJlZ2NjcwODY1MTd9.8TLs8Tazx1tpETjyCK20iCuSq94AdvMYQmDKG4nyFV8"
9- }

```

Figure 16. Routes testing from the postman

As shown in figure 17, the followers and following connections between two user with object id, So in the MongoDB database has been shown, a Testing user follow user Alamin and the Testing user Id added the user Alamin follower list, Besides, Testing user following list added the Alamin Id. These users are connected using the reference of their user id. And the posts, schools, following, and followers is an array, because a user can have many posts, schools, following, and followers.



```

registers
schools
userdatas
userdevmeets
users
test

{
  "_id": ObjectId('634c42082e939d2a4ae27fd9'),
  "posts": Array,
  "schools": Array,
  "name": "Testing",
  "email": "mdalamin.eu@gmail.com",
  "phone": 1834313691,
  "password": "$2a$10$8Jn1dC2R4HfaMtGFx0GwumtQECwvCmq9LLQ/BUvb5r.0R.Cw/LVC",
  "avatar": "://www.gravatar.com/avatar/aab02a1870935b45f968ec800de17238?s=200&r=pg&...",
  "following": Array
  > 0: Object
    _id: ObjectId('63558b850e3e0e2db1fe2f9e')
    user: ObjectId('634c4e8c2e939d2a4ae27fde')
  > followers: Array
    date: 2022-10-16T17:40:24.392+00:00
    __v: 1
}

{
  "_id": ObjectId('634c4e8c2e939d2a4ae27fde'),
  "posts": Array,
  "schools": Array,
  "name": "Alamin",
  "email": "mdalamin.ect@gmail.com",
  "phone": 1834313691,
  "password": "$2a$10$kkFc1p.2s6IWrfD.Y00U/u3CrohV01mXhFduPIV0sWl29uIkBwPmW",
  "avatar": "://www.gravatar.com/avatar/f4f8746e883c4fa1e86c2baedb952a7a7s=200&r=pg&...",
  "following": Array
  > 0: Object
    _id: ObjectId('63558b850e3e0e2db1fe2f9e')
    user: ObjectId('634c42082e939d2a4ae27fd9')
}

```

Figure 17. followers and following connections in the user object.

5.2.5 Routes

To ensure better management and clear structure of a large project, routes files must be grouped into certain folders. Three router files user, schools, and posts has been created in this project. The user route file has declared express and other modules from the different directories. For using this method from the controller, first declare `user`, `follow`, `profileController` and for the authentication, declared `authguard`.(Figure 18.)

```

routes > JS user.js > ...
 1  const express = require("express");
 2  const router = express.Router();
 3  const User=require('./controller/User')
 4  const AuthGuard = require('./middleware/authguard')
 5  const Follow = require('./controller/followersfollowing')
 6  const ProfileController = require('./controller/Profile')
 7
 8  router.get("/", (req, res) => res.send("User Route"));

```

Figure 18. Declare library and modules.

In figure 19, different routes for user modules can be seen . So, if somebody hits the route form, then he will be redirected to that specific route. In addition, the post method for the `/register`, `/login`, `/activate`, `/reset-password`, and `/profile` routes are shown in Figure 19. In addition, the Get method is used for the routes `/current` and `/myprofile` in Figure 19, while the patch method is used for the routes `enterpassword`, `edit-user`, and `experience`. The put method applies to the `/education` and the delete method applies to the `/delete-experience/:id` routes, respectively.

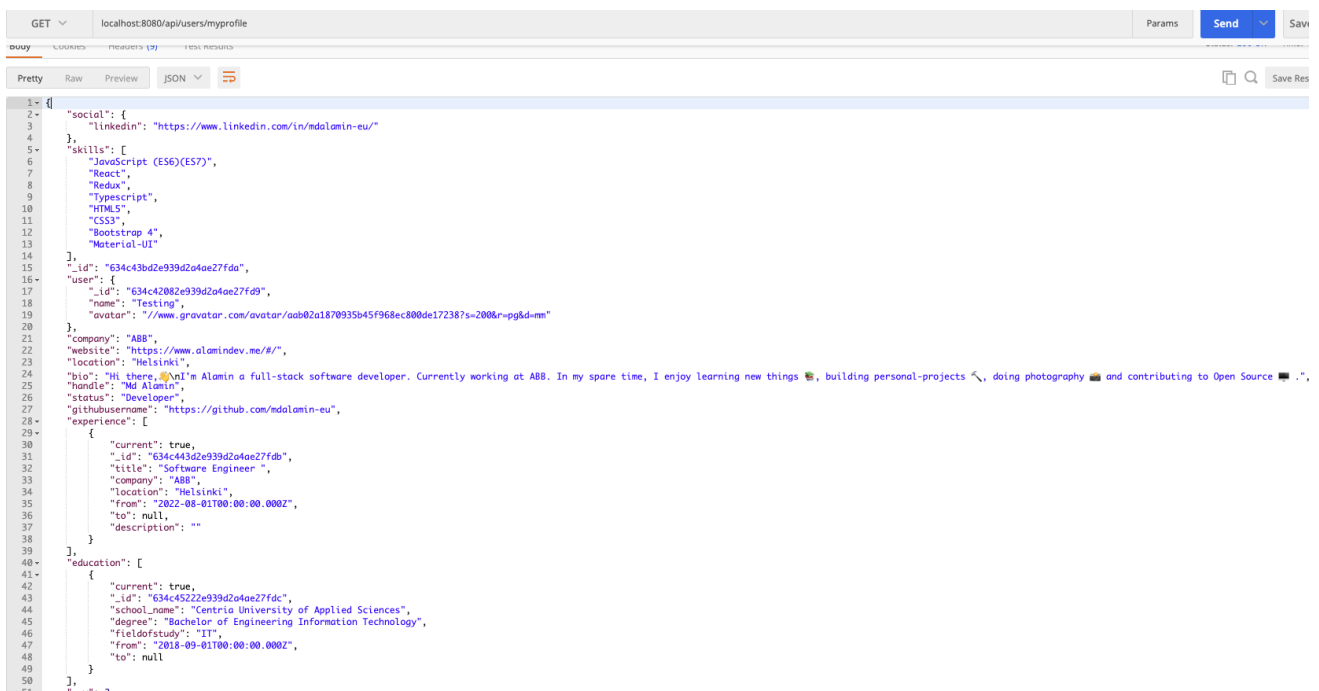
```

router.post("/register", User.registerAdd)
router.post('/login', User.Login)
router.post('/activate', User.registeractivate);
router.get('/current',AuthGuard, User.auth);
router.post('/reset-password', User.resetpasswordemailsend)
router.patch('/enterpassword', User.resetpassword)
router.patch('/edit-user', AuthGuard, User.editUserById)
router.post('/profile', AuthGuard, ProfileController.Profile )
router.get('/myprofile',AuthGuard, ProfileController.Myprofile)
router.patch('/experience', AuthGuard, ProfileController.Experience)
router.delete('/delete-experience/:id', AuthGuard, ProfileController.DeleteExp)
router.put('/education', AuthGuard, ProfileController.Education)

```

Figure 19. User routes

After hitting `/user/myprofile` and receiving a response from the server, the data are displayed in Figure 20. It contains all the logged-in user related and nested data. Figure 20 shown a Testing user details such as, user id, name, gravatar link and social account link, skills. The user entered his company name, personal website, address, bio, job title, and starting and ending dates in the experience field. The end date is null because the user is still employed by his current employer. The user entered the name of his university, degree, and field of study in the education field. The end date is null because the user is still enrolled in his current institution. (Figure 20.)



```

1- [{"social": {
2-   "linkedin": "https://www.linkedin.com/in/mdalamin-eu/"
3- },
4-   "skills": [
5-     "JavaScript (ES6)(ES7)",
6-     "React",
7-     "Redux",
8-     "Typescript",
9-     "HTML5",
10-    "CSS3",
11-    "Bootstrap 4",
12-    "Material-UI"
13-  ],
14-   "_id": "634c43bd2e939d204ae27fda",
15-   "user": {
16-     "_id": "634c42082e939d204ae27fd9",
17-     "name": "Testing",
18-     "avatar": "//www.gravatar.com/avatar/aab02a1870935b45f968ec800de17238?s=200&r=pg&d=mm"
19-   },
20-   "company": "ABB",
21-   "website": "https://www.alamindev.me/#/",
22-   "location": "Helsinki",
23-   "bio": "Hi there, I'm Alamin a full-stack software developer. Currently working at ABB. In my spare time, I enjoy learning new things 📖, building personal-projects 🛠️, doing photography 📷 and contributing to Open Source 🏠.",
24-   "handle": "Md Alamin",
25-   "status": "Developer",
26-   "githubusername": "https://github.com/mdalamin-eu",
27-   "experience": [
28-     {
29-       "current": true,
30-       "_id": "634c443d2e939d204ae27fdb",
31-       "title": "Software Engineer",
32-       "company": "ABB",
33-       "location": "Helsinki",
34-       "from": "2022-08-01T00:00:00.000Z",
35-       "to": null,
36-       "description": ""
37-     }
38-   ],
39-   "education": [
40-     {
41-       "current": true,
42-       "_id": "634c4522e939d204ae27fdc",
43-       "school_name": "Centria University of Applied Sciences",
44-       "degree": "Bachelor of Engineering Information Technology",
45-       "fieldofstudy": "IT",
46-       "from": "2018-09-01T00:00:00.000Z",
47-       "to": null
48-     }
49-   ]
50- },
51-   v: 2
52- ]

```

Figure 20. MyProfile route data.

Figure 21 has been shown all the post-related routes. If somebody hits `/post/anyotherroute` user will be redirected to that specific route. The `/addpost` route creates posts using the post method, while the `posts/:id` route uses the get method route, which only brings one post, and `/post` routes, which brings all posts. Additionally, the patch method is used to edit posts via the `/edit-post/:id` route. Finally, the Put method is used for the `/comment-post/:id` route to comment on a post and `/like-post/:id` routes to like a post.

```
6 router.post('/addpost', AuthGuard, Posts.post)
7 router.get('/:id', Posts.postById)
8 router.get('/', Posts.getAllPosts)
9 router.patch('/edit-post/:id', AuthGuard, Posts.editPostById)
10 router.put('/comment-post/:id', AuthGuard, Posts.comment)
11 router.put('/like-post/:id', AuthGuard, Posts.likes)
12
```

Figure 21. Post routes.

5.3 Frontend Creation

The frontend can be discussed at this point. This thesis discussed three core components of the MERN stack. Now is the time to explain React. But Redux JS has been used in this project. Redux provides us with client side and Redux behaves consistently across client, server, and native environments and it easy to test. Prior to this upgrade, all backend functionalities had to be tested using Postman. First and foremost, this project has installed React and Redux correctly. Additionally, React JS runs with a message in the default browser. After that, the Action, View, Store, Reducer, Routes, and Utils folders in the default-src folder were created. Frontend and backend parts are hosted from different domains. If a request is called from the client to the server, it will show an error. Therefore, we connected to the CORS using the `env` file. A CORS allows to whitelist requests from specific locations by specifying response headers like `'Access-Control-Allow-Origin'`. In cases where it is legitimate to do so, it is an important protocol to enable cross-domain requests. Additionally, it protects users from malicious websites accessing additional resources unauthorized by them. Fetching data in React and Redux JS all is a dynamic rendering using API and built most modern applications. Users can get resources from API endpoints.

```

27
28 //RegisterUser
29 export const registerUser = userData => async dispatch =>{
30   const config ={
31     headers:{
32       "Content-Type":"application/json"
33     }
34   };
35   try{
36     const res = await axios.post('api/users/register', userData, config);
37     dispatch({
38       type: REGISTER_SEND_EMAIL,
39       payload: res.data
40     });
41   }catch(error){
42     const errors = error.response.data.errors;
43     if(errors){
44       errors.forEach(error=>dispatch(setAlert(error.msg, "danger")))
45     }
46     dispatch({
47       type:REGISTER_FAIL
48     });
49   }
50 }
51

```

Figure 22. Register form and API connection.

Figure 22 shown exports a function called `registerUser`, which is an asynchronous function that sends a request to a server to register a new user with the provided user data. The function takes in an object called `userData` that contains the data for the new user, and it returns a function that takes in a `dispatch` function. The inner function is a "thunk" function that can be used to dispatch actions to a Redux store. The function first defines a `config` object, which contains a `headers` property. The `headers` property is an object with a single key-value pair, where the key is "Content-Type" and the value is "application/json". The function then uses a `try-catch` block to send a POST request to the 'api/users/register' endpoint with the `userData` and `config` as arguments. If the request is successful, the function dispatches an action with the type of `REGISTER_SEND_EMAIL` and the response data as the payload. If there is an error, the function will catch the error and extract the `errors` array from the error response. If the `errors` array is not empty, the function will iterate over the array and dispatch a `setAlert` action for each error in the array, passing in the error message and a string value of "danger" as arguments. Finally, the function will dispatch an action with the type of `REGISTER_FAIL`. (Figure 22.)

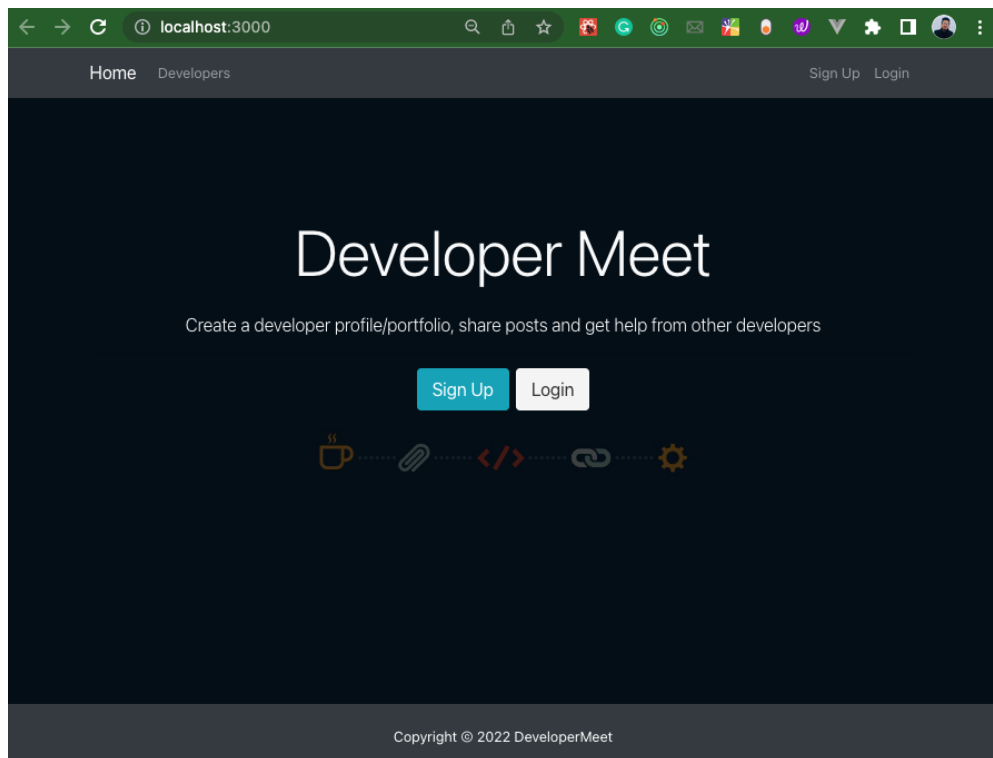


Figure 23. Frontend visualization.

At this point it is time to test the result application after creating the front-end, back-end, and connection between them. Figure 23 displays home page, which contains the signup and login page. And in the navbar user can see all developers after click Developers. To access the DeveloperMeet, users first need to create an account by clicking the Signup button using a valid email address and filling all fields. Afterwards, user can login by clicks Login button. (Figure 23.)

Figure 24. Signup.

The DeveloperMeet application's signup page with email, name, password, and confirm password is depicted in Figure 24. The DeveloperMeet user must enter a valid email address and password in all fields to sign up. In this project defines a `Register` component that is used for creating a new user account. The component has a form with four input fields `name`, `email`, `password`, `password2`. When the form is submitted, the component checks if the password and password2 values are the same. If they are, it sends a request to the server to create a new user with the name, email, and password values. If the passwords do not match, it displays an error message. If the request to create a new user is successful, it redirects the user to the `/notification` route. (Figure 24.)

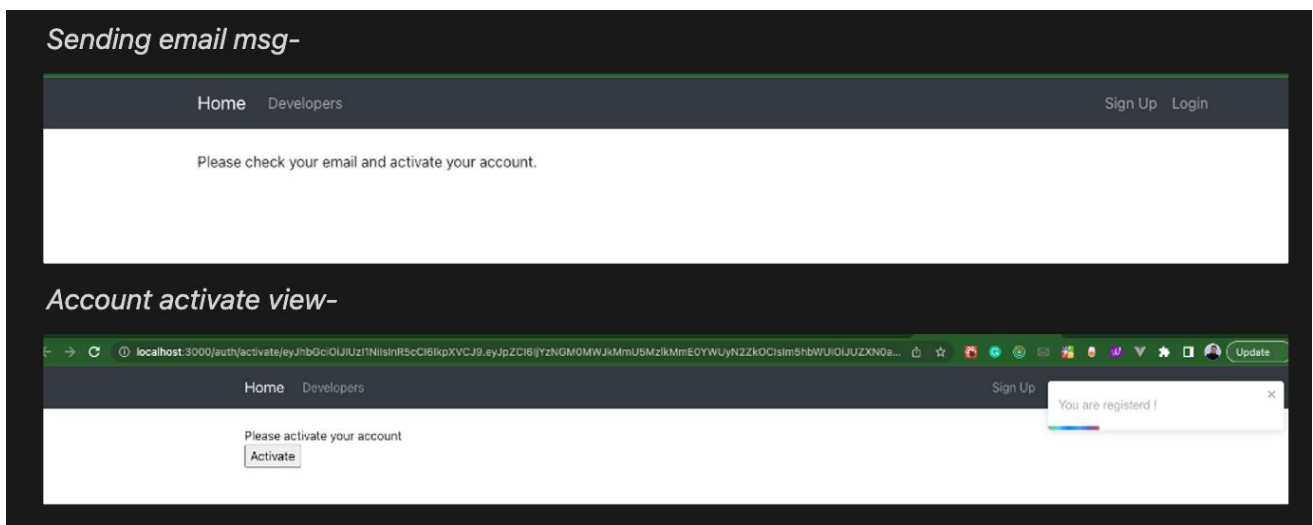


Figure 25. Sent activation email and activation page.

After submission of the registration form, user will see the message "Please check your email and active your account." And user will receive a confirmation email containing instructions on how to complete the registration process by clicking an activation link. When the user clicking this link, the activation page is displayed. Afterwards, user clicks the "activate" button it sends a request to the server to activate the user's account. The activate method is called when the form is submitted. It prevents the default form submission behavior and sends a POST request to the `'http://localhost:3000/api/users/activate'` endpoint with the token value as the request body. If the request is successful, it displays a notification using the `react-toastify` library. The notify function is called when the button is clicked. It displays a notification using the `react-toastify` library and then redirects the user to the `/dashboard` route using the `Redirect` component from the `react-router-dom` library. The render method is responsible for rendering the form and handling the rendering of the page based on the value of the token property in the component's state. If the token property is not empty, it will render a `Redirect`

component that will redirect the user to the `/dashboard` route. If the token property is empty, it will render the form. (Figure 25.)

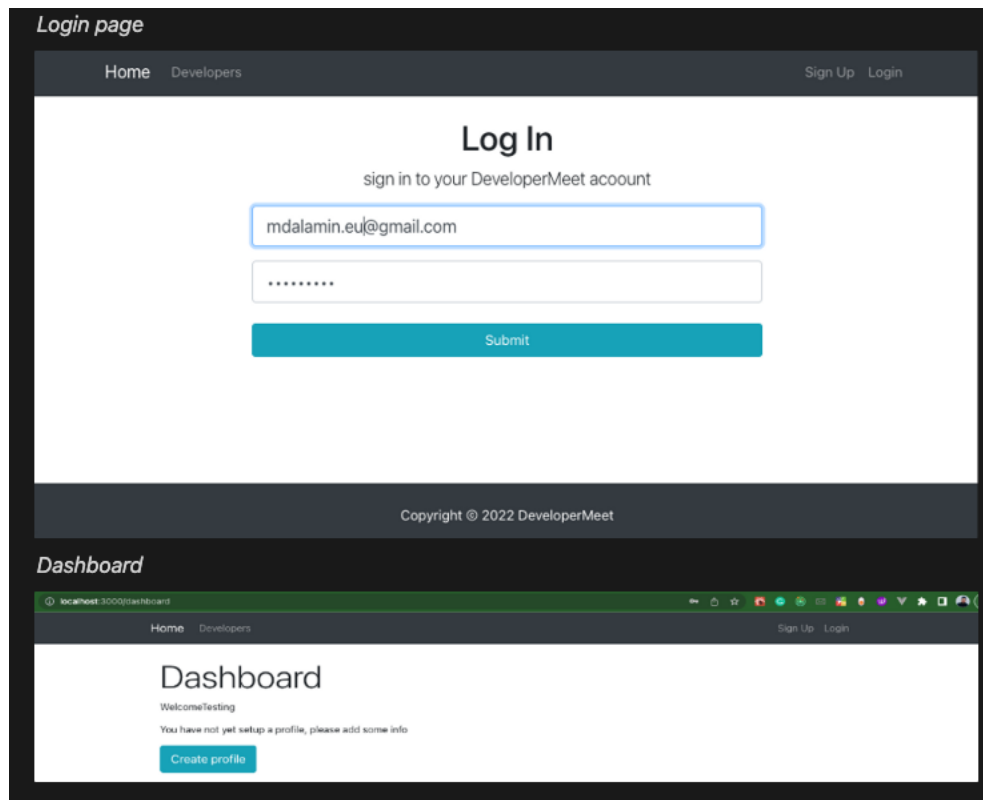


Figure 26. Login page and Dashboard.

Figure 26 shown the user can login with the email address and password and after login user can see the dashboard in the DeveloperMeet application. In addition, Figure 26 is result of `loginUser` function, the `loginUser` function takes two arguments, `userData` and `history`. Therefore, `userData` is an object containing the email and password of the user attempting to log in, and `history` is an object containing information about the current location in the application. The `loginUser` function makes an HTTP POST request to the `/api/users/login` endpoint with the `userData` object as the request body. If the request is successful, it dispatches an action of type `LOGIN_SUCCESS` with the response data as the payload and calls the `LoadUser` function to set the logged in user. It also uses the `history` object to navigate to the `/dashboard` route. If the request is unsuccessful, the function dispatches an action of type `LOGIN_FAIL` and uses the `setAlert` function to display an error message to the user. By clicking the `Create Profile` button, which takes users to a blank form on their profiles, it is now time to optimize the profile.

Home Developers Sign Up Login

Create your profile

Let's get some information to make your profile stand out

*required filled

handle
A unique handle for your profile URL example company name website profile name

* Select Professional Status
Select Status

Company Name
could be your own company or you work for one

Website
could be your own website

Location
City (eg. Helsinki)

Skills
please use comma separate values(eg, HTML, Python, Javascript)

githubusername
If you want to get your latest repos, include your github usernames

Short Bio
Tell us a little about you self

Add Social Networks Links Optional

submit

Figure 27. Create profile page.

This figure 27 code defines a React component called `CreateProfile`. The structure for completing client profiles in the `DeveloperMeet` application. The component has a state object with several properties including `handle`, `company`, `website`, `status`, `skills`, `bio`, `githubusername`, `location`. And `displaySocialInputs` like `twitter`, `facebook`, `linkedin`, `youtube`, `instagram`, `snapchat`. Profession and skills will be listed on this page. The user can move on to the next form after completing this one. The component has several methods, including `onChange`, `onSubmit`, `componentDidMount`, and `componentDidUpdate`. The `onChange` method is used to update the state when the user types into an input field. The `onSubmit` method is called when the user submits the form. It prevents the default reload behavior and destructures the state object to create a new object called `createProfileData`, which is then passed to the `createProfile` method from the component's props. The `componentDidMount` method is called when the component is mounted to the DOM, and it calls the `currentUserProfile` method from the component's props. The `componentDidUpdate` method is called when the component updates and it checks whether the profile property of the profile object from the component's props has any keys. If it does, then the user is redirected to the dashboard.

The screenshot shows a web application interface for adding education records. At the top, there is a navigation bar with 'Home' and 'Developers' on the left, and 'Sign Up' and 'Login' on the right. Below the navigation bar is a 'Go Back' button. The main heading is 'Add Education', followed by the instruction 'Add your school or degree that you have had in the past or current'. A note indicates that fields with an asterisk are required. The form contains the following fields: '*School', '*degree', 'Field of Study', 'Location', 'From Date' (with a date input field showing 'dd.mm.yyyy' and a calendar icon), a checkbox for 'Current Study', 'To Date' (with a date input field showing 'dd.mm.yyyy' and a calendar icon), and a 'Description' text area. A teal 'submit' button is located at the bottom of the form. The footer of the page contains the text 'Copyright © 2022 DeveloperMeet'.

Figure 28. User education.

In this figure 28 shows a form of education and defines a React component called `Education` that displays a table of education records. The component first declares a variable called `educationContent` that will store the JSX element to render in the table. If `education` is not empty, `educationContent` is assigned the result of mapping over the `education` array and returning a `tr` element for each record, and `tr` defines a row in a table. Each `tr` element contains several `td` elements that display the school's name, degree, field of study, and duration of the education record. And `td` defines a cell in a table. If `education` is empty, `educationContent` displays a message saying no education records have been added. Finally, the application was built for developers. After submitting this form user can proceed to the next form.

Figure 29. Experience page.

This Figure 29 defines an Experience form, in this form developers can add any previous or current job experience. Therefore, user can fill in their previous or current job/internship status. This figure 29, expects a prop called `experience`, which is an array of objects containing information about a person's work experience. The component has a single method called `render()`, which returns the JSX that will be rendered to the screen when the component is used. The `render()` method first checks if the `experience` prop is defined and has a length greater than 0. If this is true, it maps over the array of experience objects and returns a table row `<tr>` for each object, with the company name, title, and start and end dates as table cells `<td>`. If the `experience` prop is not defined or has a length of 0, the component returns a message saying, "You have not yet added your experience". The Moment component is being imported from another library called "react-moment" and is being used to format the dates in the form and to fields of the experience objects. After using the submit button to fill out the form DeveloperMeet can be used by users right away.

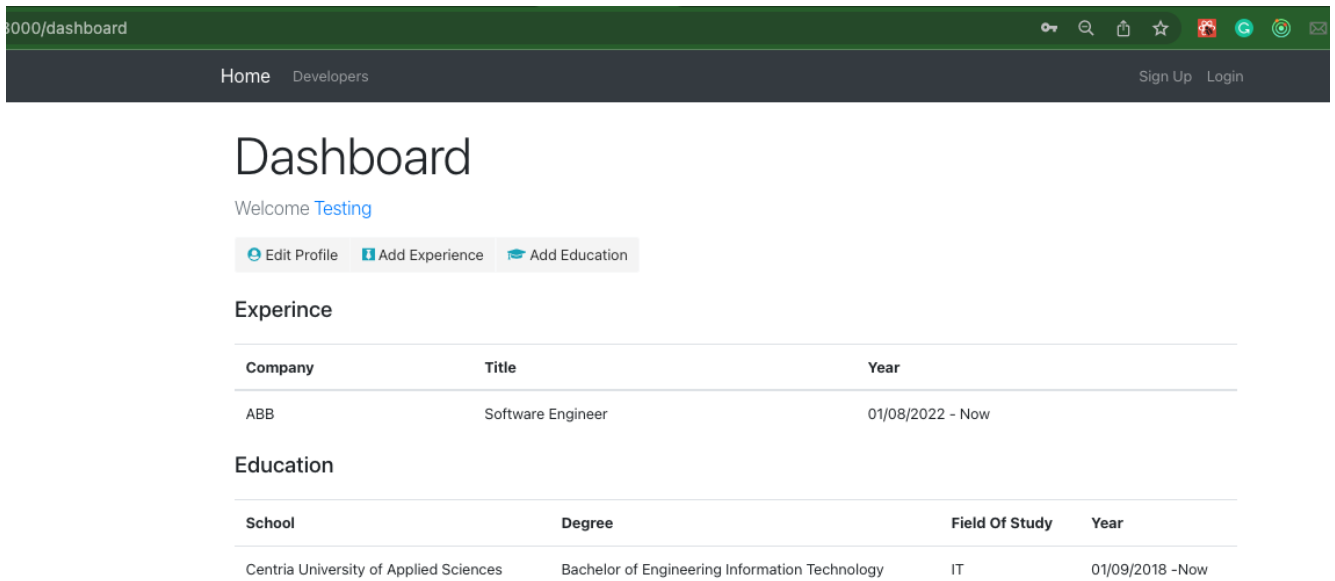


Figure 30. Dashboard.

After successfully submitting all the details from the previous parts, user can see those data in their dashboard. This figure 30 defines a Dashboard component in React, which is a JavaScript library for building user interfaces. The component is connected to the Redux store, which is a state management library for JavaScript applications, and uses the `currentUserProfile` action to fetch the current user profile data from the server when the component mounts. The Dashboard component expects two props, `profile`, and `auth`. The `profile` prop contains the current user profile data and the `auth` prop contains the current user authentication data. The component has a single method called `render()`, which returns the JSX that will be rendered to the screen when the component is used. The `render()` method first destructs the `profile` and `auth` props and the `loading` field from the `profile` prop then checks, if the `profile` prop is null, or the `loading` field is true. It returns a Spinner component. If the `profile` prop is not null and the `loading` field is not true, the code checks the `profile` object keys. If it does, it renders the username and the `Profile-Action`, `Experience`, and `Education` components, passing the `experience` and `education` fields from the `profile` object as props. If the `profile` object does not have any keys, it renders a message asking the user to create a profile and a link to the "create-profile" route.

6 CONCLUSION

Using the MERN technology stack, which includes the MongoDB database, Express JS, Node JS, and the frontend react JS, a social platform application is the aim of this thesis. Using Redux JS, bootstrap, HTML, and CSS simultaneously, this project explained the client, server, and a few frameworks and libraries. Consequently, the aim has been accomplished. This application was made to connect developers all over the world so that they can learn from each other's experiences and share their knowledge. Users can now create accounts on this platform. Additionally, they can create professional profiles. Additionally, they can acquire the skills necessary for professional success. The application is entirely new. The data will render in a few seconds and this application will not take any time to load. The thesis demonstrated that users could sign up, log in, and store their data in the database. This project built a secure authentication and authorization platform and all the minimum requirements for social media.

REFERENCES

Brown E. 2019. Express is basic framework for building web application and API with node js server. Available: https://books.google.fi/books?id=-Dq-DwAAQBAJ&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false. Accessed: 20.06.2022.

Cory G. 2015. React solves a specific set of problems, and in general, a single problem. Available: <https://itbook.store/books/9781484212462>. Accessed: 29.03.2022.

Le M. 2020. Different teams will often have very different ideas of what part of their app is the frontend. Available: <https://www.theseus.fi/handle/10024/340287>. Accessed: 02.05.2022.

Larsen R. 2013. HTML & CSS: design and build websites. Available: <https://books.google.fi/books?id=QwnWLMtXU7cC&printsec=frontcover#v=onepage&q&f=false>. Accessed: 03.05.2022.

Mardan A. 2014. NPM allows us to register our packages with a name so that we can import/export this package. Available: <https://www.amazon.com/Practical-Node-js-Building-Real-World-Scalable/dp/1430265957> Accessed: 15.05.2022.

Mardan A. 2014. Node JS is a platform for writing JavaScript applications for server side. Available: <https://www.amazon.com/Practical-Node-js-Building-Real-World-Scalable/dp/1430265957> Accessed: 15.05.2022.

Richey B. 2019. The beauty of building a modern web application is being able to take advantage of functionalities such as a Progressive Web App (PWA). Available: <https://books.google.fi/books?id=Fs6KDwAAQBAJ&printsec=frontcover#v=onepage&q&f=false>. Accessed: 22.04.2022.

Shute Z. 2019. Speed up web development with the powerful features and benefits of JavaScript. Available: <https://books.google.fi/books?id=XiWGDwAAQBAJ&printsec=frontcover>. Accessed: 02.05.2022.

Stephens R. 2008. A database can be a powerful tool for doing exactly what computer programs do best: store, manipulate, and display data. Available: <https://books.google.fi/books?id=qGgpYBighBcC&printsec=frontcover>. Accessed: 10.06.2022.

Vohra D. 2015. MongoDB consist of binary files and services that make the infrastructure of the server and store the data. Available: <https://books.google.fi/books?id=Ra1PCwAAQBAJ&printsec=frontcover#v=onepage&q&f=false>. Accessed: 25.06.2022.

Vickler A. 2021. Backend development in simple terms is all the things happening in the background that you cannot visibly see. Available: <https://www.amazon.com/Javascript-Back-End-Programming/dp/B08YFC7YZG>. Accessed: 22.04.2022.

Wilson E. 2018. The MERN stack is a collection of great tools - MongoDB, ExpressJS, React, and Node - that provide a strong base for a developer to build easily maintainable web applications. Available: <https://books.google.fi/books?id=HnxedwAAQBAJ&printsec=frontcover>. Accessed: 20.03.2022.