Bachelor's thesis

Information Technology

2020

Samikshya Aryal

MERN STACK WITH MODERN WEB PRACTICES

– Developers Connecting Application

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2020 | 44

Samikshya Aryal

MERN STACK WITH MODERN WEB PRACTICES

- Developers Connecting Application

The main focus of the thesis was to learn and develop a full-stack web application using MERN stack (MongoDB, ExpressJS, ReactJS and NodeJS) with the use of modern practices.

The thesis walks through the introduction and key concepts of MongoDB, ExpressJS, ReactJS, NodeJS along with Redux and different node packages before putting the knowledge to create the application. The process of creating the application is followed closely from its initial idea through to the end of the development and implementation phase.

As a result, a functional backend was created and tested along with a simple user interface to manage the app which was the objectives of the thesis.

This thesis acts as a brief guideline on developing a full-stack MERN web application.

KEYWORDS:

React, Redux, MongoDB, NodeJS, ExpressJS, Web Development

CONTENTS

FIGURES

TABLES

LIST OF ABBREVIATIONS

API	Application program interface
App	Application
ASP	Active Server Page
CD	Continuous Delivery
CI	Continuous Integration
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
I/O	Input/output
JSON	JavaScript Object Notation
JWT	Json Web Token
MEAN	MongoDB, ExpressJS, AngularJS, NodeJS
MERN	MongoDB, ExpressJS, ReactJS, NodeJS
MVC	Model View Controller
NPM	Node Package Manager
ODM	Object Data Modeling
PHP	PHP Hypertext Preprocessor
REST	Representational State Transfer
SQL	Secure Socket Layer
UI	User Interface
VCS	Version Control System
XML	Extensible Markup Language

1 INTRODUCTION

Web technology is growing tremendously as there have been significant developments in the past few years which have made developers work easier and more convenient. When the Internet began in 1989, many developments and changes have occurred in web technologies. In the first years of the Internet in 1993, the web was introduced and is called HTML (Hyper Text Markup Language) (Jacksi and Abass, 2019). After a few years later in 1996 CSS (Cascading Style Sheets) were introduced as a means of styling web pages. However, both technologies are still in practice on every web page today. Today, developers are provided with so many services and options to play around web technologies. While in the past HTML, CSS, JavaScript was used as options for web-design. Many frameworks and libraries have been developed to create a web page following certain patterns. In the end, all of them are focused to develop good looking, reliable and functional web applications.

However, due to the availability of many options, choosing an appropriate stack leads to confusion. Out of different stack available today, MEAN (Mongo DB, Express JS, AngularJS, and Node) and MERN (Mongo DB, Express JS, ReactJS and Node) are the most popular stacks around JavaScript. Both stacks are open source and offer an end-to-end framework to build comprehensive web apps that enable browsers to connect with databases. Any syntax errors or any confusion can be avoided by just coding in one programming language, JavaScript. Another advantage of building web projects with MEAN or MERN is the fact of its enhanced flexibility (Bhardwaj, 2018).

The thesis will explain the process of development of full-stack web application with best practices of MERN stack such as reusability of JavaScript components, usage of React hooks, MVC pattern (Model View Controller) and many more from scratch. The thesis starts with a brief introduction of each tech Stack and then proceeds with the detailed implementation of each of the Stack processes. A prototype of an application is developed to demonstrate the implementation of different forms of JavaScript being used to form a single page web application.

2 INTRODUCTION TO FULL-STACK WEB DEVELOPMENT WITH JAVASCRIPT

A full-stack is a combination of front-end and back-end. Front-end generally refers to the portion of the application the user sees or interacts with, and the back end is the part of an application that handles the logic, database interactions, user authentication, server configuration, and so on. A full-stack application can be in any form like a web application, mobile application, or some other application. A web application is a creation of application programs that are delivered to the users' devices over the use of the internet. The web application is accessed through a network and does not need to be downloaded. An end-user can access a web application through web browsers such as Google Chrome, Safari, or any other browsers.

Web technology is growing tremendously, and there has been a creation of different stack for developing a full-stack web app. Among them, JavaScript is one of the dominant programming languages in web development, that has been around for over 20 years. In the mid-2000s, the shift from websites to web applications, along with the release of faster browsers. JavaScript developers created libraries and tools that shortened the development cycles, giving birth to a new generation of even more advanced web applications which in turn created a continuous demand for better browsers. The real revolution began in 2008, when Google released its Chrome browser, along with its fast JIT- compiling V8 JavaScript engine. Google's V8 engine made JavaScript run so much that it completely transformed web application development (Haviv, A., 2014).

Since JavaScript supports both client-side and server-side, it has several benefits over choosing different languages for client-side and server-side. With the knowledge in a single programming language, JavaScript one can build a full-stack web application. It is a good stack for developing a dynamic and high performing application. The JavaScript codes are reusable, they have good tutorials, they are relatively easy to learn and implement, numerous resources can be found in Stack Overflow and GitHub projects, they offer faster development, great distribution through npm. Similarly, JavaScript versions are updated annually by ECMA (ECMA International – European Association for Standardizing Information and Communication) (Haviv, A., 2014).

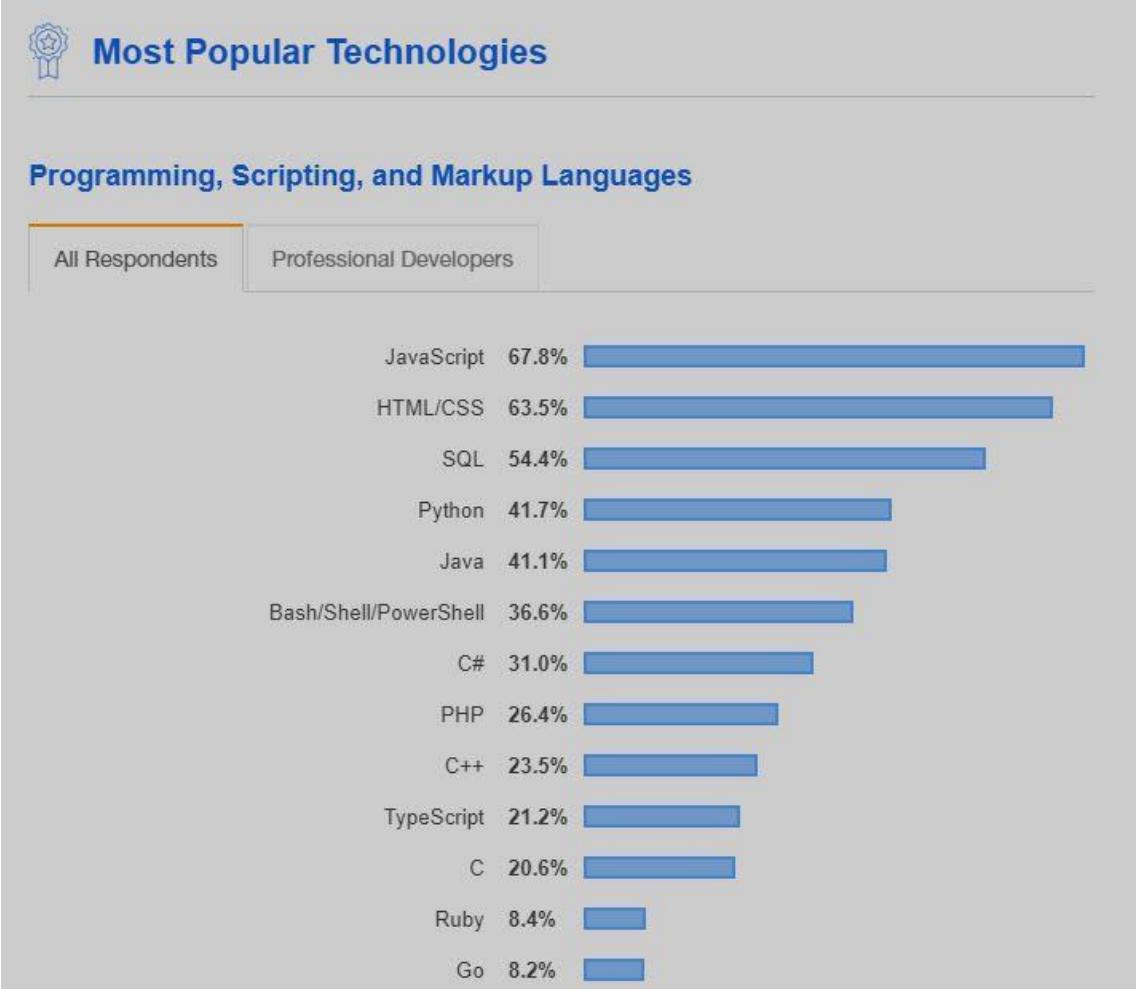


Figure 1. Stack Overflow Developer Survey Results from 2019 (Stack Overflow Developer Survey 2019, 2020).

As illustrated in Figure 1, 67.8 percent of the total surveyed listed JavaScript as the most used technology. This clearly shows that JavaScript is one of the most popular programming languages. Languages like PHP, ASP, Ruby, and Python are among the early server-side languages. Lately, JavaScript has made possible to create full-stack combining all the factors like a client-side user interface (UI), server-side communication, server, and database to interact to make a complete package for web development.

MEAN and MERN are the most common stacks of JavaScript today. Both stacks are similar only the front-end portion is different i.e. React library and Angular framework. As React, is one of the popular among single page front-end libraries of JavaScript that makes rendering part of the page easier without refreshing the page hence was chosen for the thesis.

COMPARISON TABLE

S.N	Attribute	MEAN	MERN
1	DOM	Regular DOM	Virtual DOM
2	Learning Curve	Weak	Strong
3	Packaging	Weak	Strong
4	Abstraction	Weak	Strong
5	Debugging General	Good HTML / Bad JS	Good JS / Bad HTML
6	Debug Line NO	No	Yes
7	Unclosed Tag Mentioned?	No	Yes
S.N	**Attribute**	**MEAN**	**MERN**
8	Fails When?	Runtime	Compile-Time
9	Binding	Two Way	Uni-Directional
10	Templating	In HTML	In JSX Files
11	Component Model	Weak	Medium
12	Building Mobile?	Ionic Framework	React Native
13	MVC	Yes	View Layer Only
14	Rendering	Client Side	Server Side

Figure 2. Comparison Table of MERN and MEAN stack (Bhardwaj, 2018).

As in Figure 2, React uses virtual DOM while Angular uses Regular DOM similarly, the templating of the React is in JSX whereas the Angular has in HTML. The MERN stack was chosen due to its popularity for having an advantage in the above-mentioned table attributes. Therefore, the thesis is focused on developing a full-stack using MERN stack.

3 TECHNOLOGIES AND CONCEPTS

As briefly discussed above MERN stack consists of four independent frameworks and libraries, MongoDB, ExpressJS, ReactJS and NodeJS which supports MVC architecture to make the development process flow smoothly. MongoDB plays a role for database management, while NodeJS and Express are used for building routes and APIs in the backend and ReactJS is used in the frontend. The connection between them is made (The Modern Application Stack – Part 1: Introducing The MEAN Stack | MongoDB Blog, 2020). Each of the technology is used to develop an application and each technology will be explained in deep in the coming chapters.

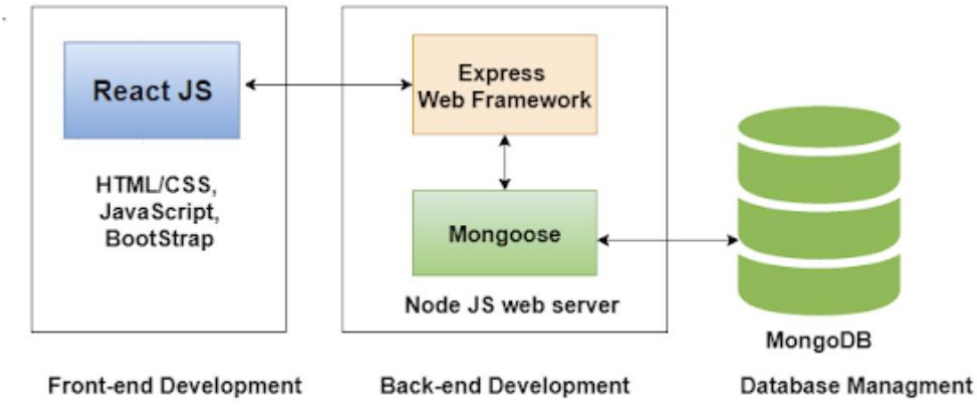


Figure 3. Full-stack MERN architecture.

3.1 Front-end Framework

3.1.1 ReactJS

React is one of the most widely used open-source, declarative, efficient, and component-based JavaScript libraries that helps to build interactive UI where all the components are reusable and can be applied for multiple projects. It was developed by Facebook's engineer and is continuously handled and updated by Facebook and the users around the globe. React is mainly concerned with re-rendering data of the DOM. It requires additional libraries for the management, route handling and API interaction.

Comparing with another front-end framework i.e. Angular, React uses views that make code more predictable and easier to debug. It also helps to update DOM efficiently and render the right components when the data changes.

```
class HelloMessage extends React.Component {
    render (){
      return (
       <div>Hello {this.props.name}</div>
    );
  }
}
 ReactDOM.render(<HelloMessage name="John"/>,
    document.getElementById("mydiv")
);
```

Figure 4. A simple React Component.

The above snippet Figure 4 implement a render() method that takes input data and renders the display. A div tag with the content of plain text "Hello, John" will appear in a DOM-element with the id "mydiv". The above code uses an HTML like syntax called JSX which is a syntax extension to JavaScript. Input data that is passed into the component can be accessed by render () via this.props() (Elboim, 2017).

React components are small, reusable pieces of code. They return a React element to be rendered to the page. It is also a normal JavaScript function that takes input or props which is mean to be properties.

Components can return other components, arrays, strings, and numbers and, can be either functional or class-based, with or without states. Normally, functional components do not have any states in them while stateful components can holds states and pass on properties to the child-components. Class components can have lifecycle methods and hence they can make changes to the UI and data. There is a rule that is applied for writing component names. The component name should always start with a capital letter (Elboim, 2017).

React Hooks

Hooks were introduced to React in version 16.8 that enable the execution of custom code in a base code. Reacts Hooks are special functions that allow to “hook into” its core features. It provides an alternative to write class-based components by allowing to handle state management from functional components.

Even though component-oriented architecture allows reusing views in the application, one of the biggest issues a developer meets us how to use the logic located in the required component state between other components. Where there are components with similar state logic, but no good solutions to reuse the logic, the code in the constructor and life cycle methods can be duplicated. To fix this developer usually use high-order components and render props (Introducing Hooks – React, 2020).

Hooks are aiming to solve the issues allowing writing functional components which have access to state, context, lifecycle methods, refs etc. without writing React Classes (Introducing Hooks – React, 2020). An example of React hooks can be observed in Figure 5.

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Figure 5. Example of usage of React Hooks (Introducing Hooks – React, 2020).

In the above code, the class Component is replaced with a functional component, the local state is removed and instead the new “useState” function is used.

Hooks are used in functional component to use state, useState() hook is used, while to replace life cycle methods like componentDidMount() or componentDidUpdate(), useEffect() hook is used.

3.1.2 Redux

As the data is stored in states among components in React, handling big scale application data becomes difficult and gets complicated. These states can include server responses and data, as well as locally created data. UI state is also increasing in complexity, so as we need to manage active routes, selected tabs, spinners, pagination controls and so on. To tackle all the big complications Redux is applied (Redux, 2020).

Redux is composed of 3 core principles; state to store necessary data, action to dispatch changes made to the state and a reducer to make changes. When a user creates an event, actions are dispatched to the central store. Actions are the only information that the store receives from the user. This makes reducers fire a method on the previous state depending upon the action received from the store. It also decides the fate of the old state.

App's state is mainly described as a plain object. For example, the state of people is presented below.

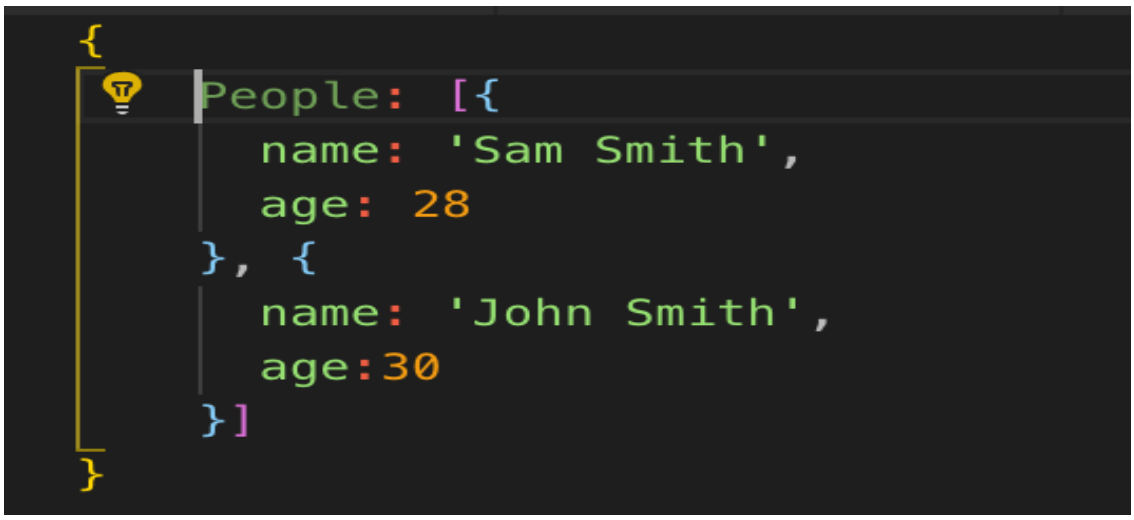


Figure 6. Example of state in Redux.

As states in redux are changing, an action is dispatched containing the payload and reducer takes the role of making the changes to our state. All the state in an application is stored in Redux as a single object. Reducers take an action and decide what method to be implemented or deleted depending upon the action that is being dispatched. This is the only way the new state gets produced every time (Redux, 2020).

```
// action
function addTodo (text) {
    return {
        type: "Add_TODO",
        payload : text
    }
}
  // reducer
  function todos (state = [ ], action){
      switch (action.type) {
          case 'ADD_TODO':
            return [...state, action.payload];
        default:
            return state;
    }
}
```

Figure 7. Example of action creator and reducer in Redux.

After the state is changed Reducer will not alter the state directly. A copy of the state is created, and a new person is added to the state. This principle of the redux is to directly avoid the mutation of the previous state (Redux, 2020).

3.2 Back-end Framework

3.2.1 NodeJS

Node.js is the currently most popular open-source web server environment that allows JavaScript code to be able to run outside of web browsers. It creates a dynamic web

page by helping JavaScript to write command-line tools and server-side scripting before the page is sent to the user's web browser (Express/Node introduction, 2020). Node.js is suitable for the foundation of a web library or framework because it has an HTTP a first-class citizen in Node.js, designed with streaming and low latency. It uses a Google Chromes v8 engine to execute the JavaScript. NodeJS, in general, makes developing cross-platform, server-side and networking applications much easier (Chettri, 2016).

Node has earned a good reputation in the tech industry in a short period. It plays a big role in the technology stack of high-profile companies who depend on its unique benefits. Microsoft Azure users are provided with end-to-end JavaScript experience for the development of a whole new class of real-time applications. The locking and concurrency issues of eBay were freed by Node's multi-threaded asynchronous I/O. The entire mobile software of LinkedIn is completely built-in Node (Martonca, 2015).

PayPal after shifting to Node from Java for their existing projects saw significant improvements over Java. The re-written app using Node was delivered in half the time with less manpower and fewer lines of code. Hence, PayPal saw their development and product performance increase dramatically after switching to Node from Java (Anderson, 2014).

One great advantage of NodeJS is the event-driven architecture which allows program code to be executed asynchronously. The most common tasks for servers include answering queries, storing data in a database, reading files from hard drive, and establishing connections to other network components. All these actions are grouped together under the category I/O. In a programming language like C and Java, I/O is executed synchronously. This means that tasks are performed one at a time and moves to the next task after completion of the first task. However, Node uses an asynchronous I/O, with which read and write operations can be delegated directly to the operating system or database. This makes possible for many I/O tasks to be carried out parallel to one another, without any blockage. In some cases, this can prove to be a great advantage when it comes to the speed of the NodeJS and JavaScript-based applications.

NodeJS also provides tools and functions like NPM and "require", that help to manage third party libraries like Mongoose and Express to make the development faster and efficient. NPM is the default package manager which comes with the Node environment. Such tools help to load inbuilt node modules into the applications (Chettri, 2016).

```
const http = require('http')

const app = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/plain' })
  res.end('Hello World')
})

const port = 3001
app.listen(port)
console.log(`Server running on port ${port}`)
```

Figure 8. Simple web server.

In the above code, a request is made to the server. When the code runs, the message "Server is running on port 3001" is displayed in the console. When visiting the browser address http://localhost:3001: "Hello World" message is displayed (Fullstack part3 |, 2020).

3.2.2 ExpressJS

Express is a Node.js web application framework that boosts the features of web, mobile API applications, and APIs. Many libraries have been developed to ease sever side development with Node, by far the most popular intended library for this purpose is express. The third-party middlewares are installed to extend the functionality of the express application and add features as required (Fullstack part3 |, 2020).

In Express middleware is executed from top to bottom in the request-response cycle. Each middleware has access to the request and the response object as well as a next function which are passed from one middleware to another. The middleware takes the request, executes the code inside, changes the request and response objects and calls the next function which activates the next middleware in the queue. An express application can have access to the application-level middleware, router-level middleware, error-handling middleware, built-in middleware, and third-party middleware. In application-level middleware app.use() and app.method() functions are used where 'method' is an HTTP verb that is being executed. The router-level middleware is bound to an instance of the express router. Error-handling middleware is around the object and takes four arguments, error, request, response, and next objects. "Express.static" and

“express.json” files are served statically and parse the incoming request to JSON respectively in built-in middleware. Different functionality of the express application can be extended either by logging in the information about the incoming request or parsing the cookies (Fullstack part3 |, 2020).

Furthermore, routers in express divide the application into several mini express applications and combine to form an express application. The express server is composed of an HTTP verb (GET, POST, PUT, DELETE). The same server creation from the above code (Figure 7) can also be done with express. In express callback functions are specified to the routing where callback functions can be used as arguments. Callback functions are called by using the next() method to move on to another callback function.

```
const express = require('express')
const app = express ()
app.get('/', (request, response) => {
  response.send('<h1>Hello World!</h1>')
});
```

Figure 9. Example of a server built with Express.

This thesis demonstrates more about express in the development phase for adding different functionalities like building the routes APIs, use of all the HTTP verbs and for user authentication.

3.3 MongoDB and Mongoose

3.3.1 MongoDB

MongoDB is an open-source, cross-platform database that encompasses a wide variety of different database technologies designed to fit into modern applications. MongoDB uses JSON like files schemas, meaning data are stored as documents or collections which can be strings, numbers, floats and even arrays and objects. MongoDB is also known as NoSQL database; it allows the insertion of data without interrupting the service

and provides the real-time application changes. Due to the vast complexity in web development and usage, the relational database cannot cope with the scale and agility challenges of modern applications. When comparing relational database to NoSQL, the NoSQL database has better performance and is more scalable. In this thesis, there is the use of the NoSQL database, which is MongoDB for the fast development, code integration and for less database administrator time (MongoDB System Properties, 2020).

Despite having so many advantages, MongoDB also has some drawbacks, as MongoDB stores key names for each value pairs due to no functionality of joins there is data redundancy which results in unnecessary consumption of memory. There is a limit for document size not to exceed 16MB. Also, the nesting of documents is not allowed to have more than 100 levels. However, the ability to store any kind of data without structuring any rule, governing the relation can be a strength at times. Applications can be built using any platforms and provide the preferred data-structure (MongoDB System Properties, 2020).

3.3.2 Mongoose

Mongoose is a query language to communicate between the server and the database and performs reading, writing, updating and deleting operations of the data. It is an Object Data Modeling (ODM) library for MongoDB and NodeJS which provides schemas to model application data which therefore cleans up the ambiguity of databases. It enforces a standard structure to all the documents in a collection using schema. It also validates the stored data in the documents as well as allow only valid data to be saved in the database. Overall Mongoose provides all the features to the MongoDB including query building functions and business logic in the data (Mongoose ODM v5.9.12, 2020).

4 PROJECT IMPLEMENTATION

Developers Connecting Application (DEVAPP) is a web application project designed using the MERN stack which facilitates the user to create a resume profile and other features. This thesis guides the process of making a managed resume of developers who are looking to create a profile. The sole function of the app is to help the developers create a nice-looking profile. There is a dashboard where the users must sign up first and later, they can log in and create own profile. In the dashboard, a normal user without registering into the application can view the profile of users who have already created the resume in the system. The app has features like view profile, edit-profile, add-education, and add-experience. Also, users are facilitated with the options to delete their profile permanently. By creating a profile user can give and take the feedback on own's and other developers profile. Users from anywhere can be connected by resume. Also, this thesis could be a good social networking site like other social sites. These are the only feature the app included for now and other features or improvements will be considered if the application continues to be developed. The architecture of the project is shown below.

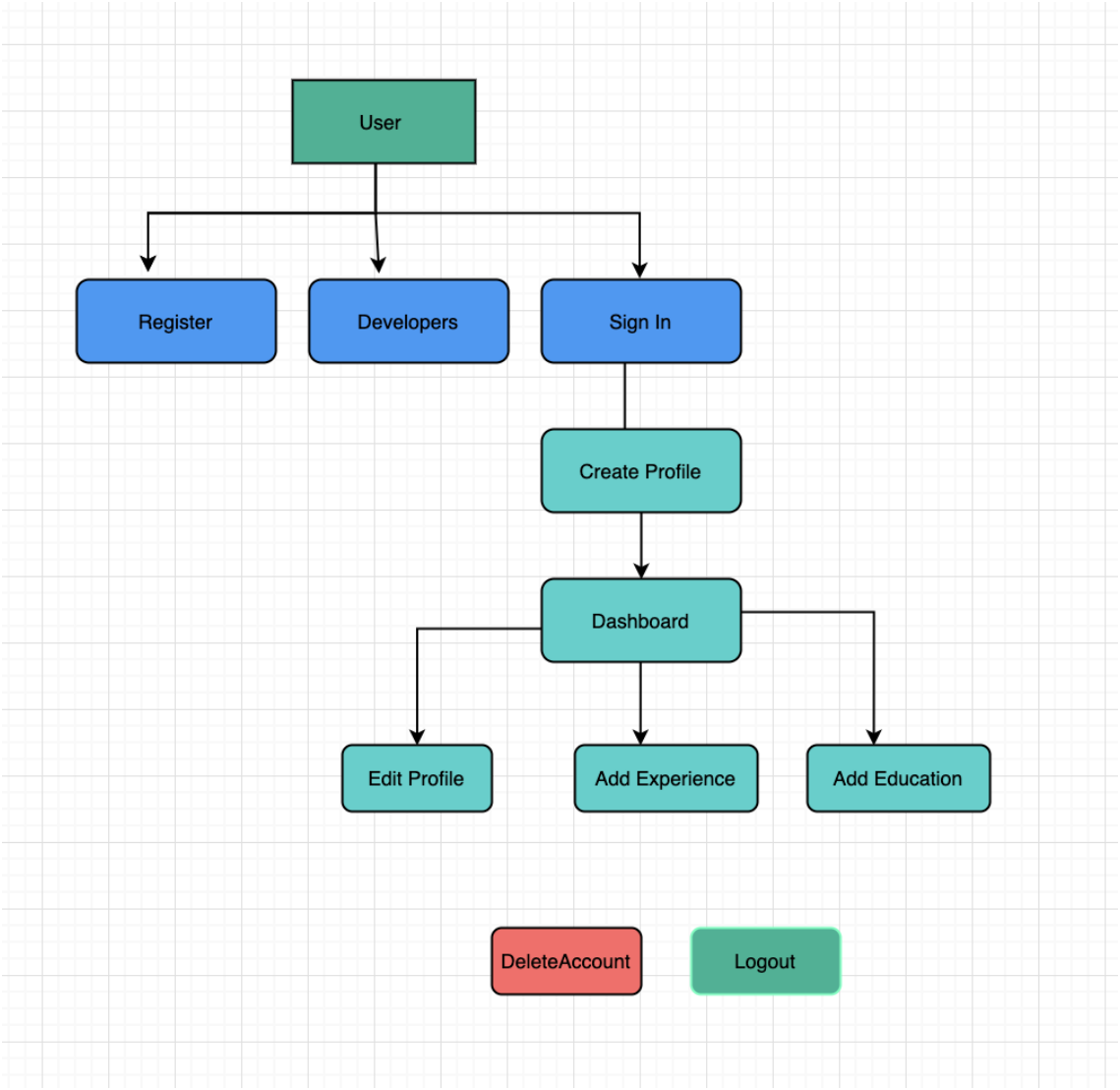


Figure 10. The architecture of the project.

4.1 Development Environment Setup

It is always important to choose the right tools and a suitable environment to perform any kind of task. A combination of such tools and environment increases the efficiency of work, increases productivity, and saves a lot of time. Choosing a comfortable stack is also one of the keys to succeed in the development process and make it easier.

The application was entirely developed in MacBook Pro 2018 with macOS Catalina 10.15.3 as the operating system. Besides, several third parties' tools and software were used to make the development process easier and get an expected result.

Visual Studio Code was used as a code editor for this project. VS Code is an open-source project spearheaded by Microsoft, which was first released in 2015. Since then it has become one of the most popular code editors among the developers. VS Code is a great package starting from setup, adopting UI for showing the content of files, customizable settings, debugging a NodeJS web app to the deploying of the web application to the cloud (Code, 2020).

4.1.1 Version Control System (VCS)

Version control is a component of software configuration management, that keeps a record of changes to a document or set of documents over time so the specific versions can be recalled later. Changes are identified by a number or letter code, which is termed as “revision number”, “revision level”, or simply “revision”. For instance, an initial set of files is “revision 1”. When the change is made, the resulting set is “revision 2” and so on. “Each version is associated with a timestamp and the person making the change. Revision can be compared, restored, and with some types of files, merged” (Blischak, Davenport and Wilson, 2016). Version control makes a group or team working together on a project easier to collaborate. There are many different version control systems available which are Git, SubVersion, Team Foundation Version Control and so on. Git one of the most common version control system and was chosen for the project as this is a distributed version control system which emphasizes on speed, data integrity and support for distributed non-linear workflows (Blischak, Davenport and Wilson, 2016).

The advantage of Git is that it provides backups by the nature of its design. Whereas other VCSs have a ‘single source of truth’, every Git repository stores the entire repository. Typically, developers have a working copy of a repository which is stored locally on their computers, as well as one repository stored in a Git cloud provider such as GitHub, GitLab or BitBucket.

4.1.2 Framework Installation and Node Packages

Node Package Manager (NPM)

NPM is a package manager for the JavaScript which is included as a recommended feature in Node.js installer (About npm | npm Documentation, 2020). It also shares and

borrows packages by developers, developers can reuse other work which normally comes in packages. Each package is a collection of codes that are maintained and managed by a package management system. NPM also installs and updates the packages automatically. NPM consists of three distinct components: the website, the command-line interface (CLI) and the registry. Developers use the website to discover packages, CLI runs from terminal to interact with NPM and registry is a large public database of a JavaScript software and the meta-information surrounding it. NPM is the default NodeJS package manager.

NodeJS

The NodeJS version 12.13.1 was installed from the official website in the macOS. The application was initialized using the command "*npm init"*. The name of the application, version, and description of the project was specified during the installation. A file name "package. json" was generated automatically with the information provided during the installation.

ExpressJS

After installing NodeJS, Express framework version 4.17.1 was installed globally with the command "*sudo npm install -express -save -g"*.

MongoDB

MongoDB can be installed locally to the machine the application is being developed on or directly to the MongoDB Atlas as a cloud solution. An Account was created in MongoDB website. A free MongoDB cluster was set up after creating a new project. Apart from this, read and write access to the database was created which was used in the NodeJS application to read and write the data from and to the database. Also, the current IP address of the machine was added to the IP whitelist so that the app can communicate with the server. Finally, the MongoDB driver was installed in the application using the command "*npm install mongodb –save"*.

The overview of a MongoDB cluster for the application is presented below.

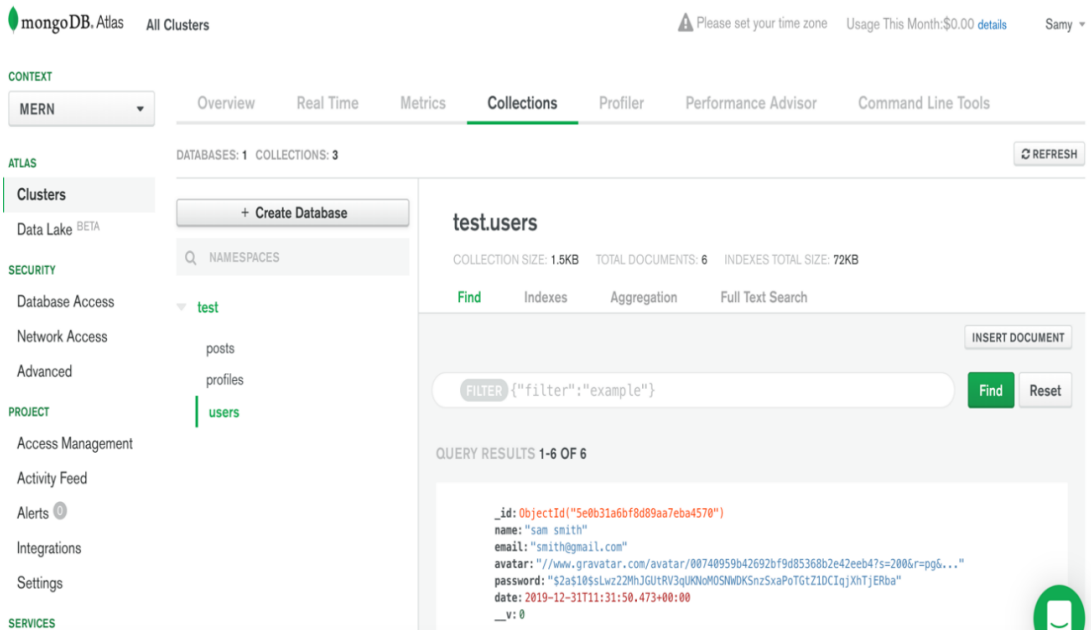


Figure 11. A screenshot of MongoDB Atlas of the application.

Mongoose

The further step is to make use of mongoose to make server and database communicate with each other. The mongoose was installed using the *command npm install mongoose –save* to get the latest version, the version available at the time of installing was 5.8.1. The database connection was created between the application and the server using the URL, username and the password created while setting up the MongoDB Atlas. Besides, the required schemas and models were created which will be discussed later in chapter 4.2.2.

ReactJS

The React version installed during the process of development of the project was 16.12.0. As React represents the abstraction of client-side used, it was used to develop a user-friendly interface of the application. Also, to reduce the stress of doing CSS styling from scratch and to re-use all the components within the project React was used.

Along with React, Redux was used to manage the states as the application is big enough to and states were difficult to handle only with React. React Hooks were also used to abandon the use of Class Component in favour of Functional Component.

Besides, other node packages were installed in the project to provide more features and make the development process easy. Some of them are discussed below.

Nodemon concurrently

Nodemon concurrently is a tool that helps to run more than one command inside the terminal. This tool is handy when running front end and back end of the project simultaneously (concurrently, 2020).

Bcrypt

Bcrypt package was installed using the command "*npm install bcryptjs".* This package was used to store a password into the database in a secured way. Bcrypt has two methods, Salt hashing and Gen hashing to hash the password (bcryptjs, 2020).

Express Validator

Express Validator is one of the many npm packages for validating a request of an express application. It was installed into the application by using the *command "npm install express validator"* (express-validator, 2020).

4.2 Application Logic

In the development phase, all the tools and packages were installed in the beginning. After the initial setup of the application, actual coding was started. Below is the snippet of the final files and folder structure of the application.

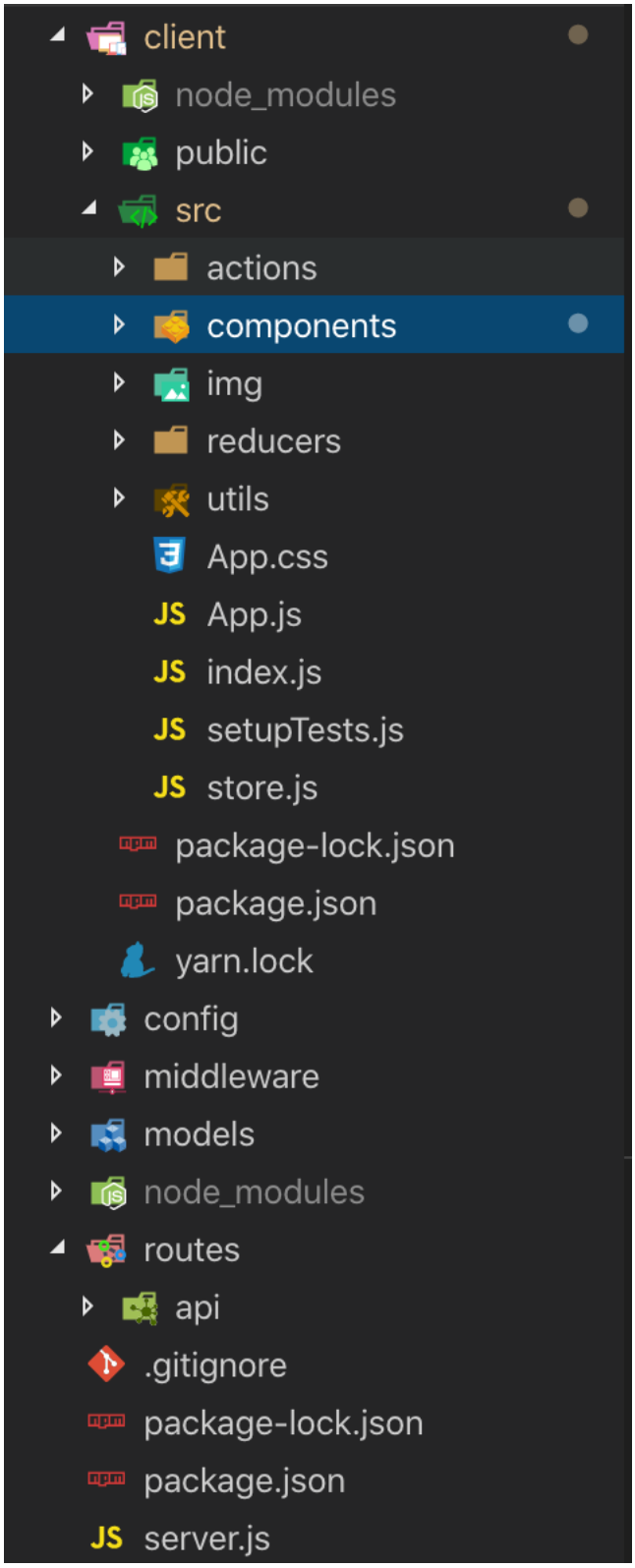


Figure 12. The files and folder structure of the application.

In the above folder structure of an application, there are two folders “client” and “routes”. Client folder contains all the files and folders of the front end while routes folder contains all the files and folder of backend. The main source code is located under the “src” folder from the default, auto-generated ReactJS “node_modules”. The entry point of the project is in the “App.js” file where the initialization of the global state is happening. The role of each of the folder will be discussed more later.

4.2.1 Authentication Handling

Authentication is the process of verifying the identity of a user who is accessing the application. Users can verify themselves to the server with the simple login form, and also with various identification system like fingerprint, voice, face recognition or by retina scans. However, the most common, simple, and cheap form for verifying the authenticity of a user in web applications is via the login form. All the authentication in this app is handled by JWT. In the beginning, the user has to create an account with some basic information like email and password which is encrypted, and the information is saved in the database. Each time user logs in using their credentials, a JWT is sent back. Every API call from the client-side requesting permission to access the protected route must have an Authorization header with a JWT attached, which gives the data of the person that makes the request and checks if they have permission (JWT.IO - JSON Web Tokens Introduction, 2020). This is facilitated by custom-Middleware folder in the root, that intercepts the request and checks if it passes the conditions to receive the response from the server. If there is no token at all, the route is protected using middleware and an error message pop up in the screen. Also, if there is a token and is not valid, again error message is thrown. And if the token is valid, it is decoded, and it is set in a req.user and can be used in any of the protected routes. The session is destroyed once the user logs out of the application or when the browser is closed (JWT.IO - JSON Web Tokens Introduction, 2020).

```
const jwt = require('jsonwebtoken');
const config = require('config');
module.exports = function ( req, res, next) {
    const token = req.header('x-auth-token');
    if(!token) {
        return res.status (401).json({msg: 'No token, authorization denied'});
    }
    try{
        const decoded = jwt.verify(token, config.get('jwtSecret'));
        req.user = decoded.user;
        next();
    } catch (err) {
        res.status(401).json({msg: 'Token is not valid'});
    }
}
```

Figure 13. Validation of JWT

4.2.2 Models

Models were created to define a standard structure for the documents being stored in a specific model. The application mainly consists of two models, Profile and User. The profile model and the user model store the collection of profile documents and user documents, respectively. Below is the user model and is exported to be used in the other parts of the application.

```
const mongoose = require('mongoose');
const UserSchema = new mongoose.Schema({
    firstname: { type: String, required: true},
    lastname: { type: String, required: true},
    email: { type: String, required: true,  unique: true },
    password: { type: String,  required: true },
    avatar: {  type: String, required: true},
    date: { type: Date, default: Date.now}
});
module.exports = User = mongoose.model('user' , UserSchema);
```

Figure 14. User model.

In the above figure, an instance of mongoose's schema is formed and named userSchema. The schema represents the structure of a single document in a collection.

The example of JSON with different keys that the user has filled when signing up in the app is presented below.

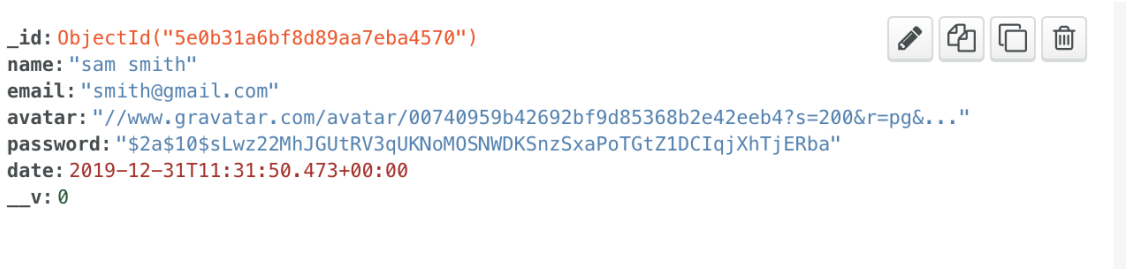


Figure 15. Example of user document in JSON.

After successful registration, users can create a profile using their credentials and the profile is saved into the database. The profile model is presented below.

```
const mongoose = require('mongoose');
const ProfileSchema = new mongoose.Schema({
    user:{ type: mongoose.Schema.Types.ObjectId, ref:'user'},
    company:{ type: String },
    website:{ type: String},
    location: { type: String},
    status:{ type: String, required: true},
    skills:{ type: [String], required: true},
    bio:{ type: String },
    githubusername:{ type: String},
    experience:[{
            title: { type: String, required: true},
            company:{ type: String, required: true},
            location:{ type: String },
            from:{ type: Date, required: true},
            to:{ type: Date },
            current:{ type: Boolean, default: false },
            description: { type: String }
        }
    ],
    education: [{
            school: { type: String, required: true},
            degree:{ type: String, required: true},
            fieldofstudy:{ type: String, required: true },
            from:{ type: Date, required: true },
            to:{ type: Date},
            current:{ type: Boolean, default: false },
            description:{type: String}
        }
    ],
    social: {linkedin :{type: String} },
    date:{type: Date, default: Date.now
    }
});
module.exports = Profile = mongoose.model('profile', ProfileSchema);
```

Figure 16. Profile model.

The profile model has a similar pattern as the user model but contains more keys. Keys like a user, company, website, location, status, skills, bio, GitHub username, experience, education. When creating a profile, the user must fill all the keys. The user model is then exported to the backend in the file called profile.js. Different routes were created in the profile.js which will be elaborated in the coming chapter.

The user must fill all the keys in the front end to create a profile. The JSON is then saved in the database. Keys like skills, experience, education and GitHub username are in the Array form. JSON of the user profile is presented below.

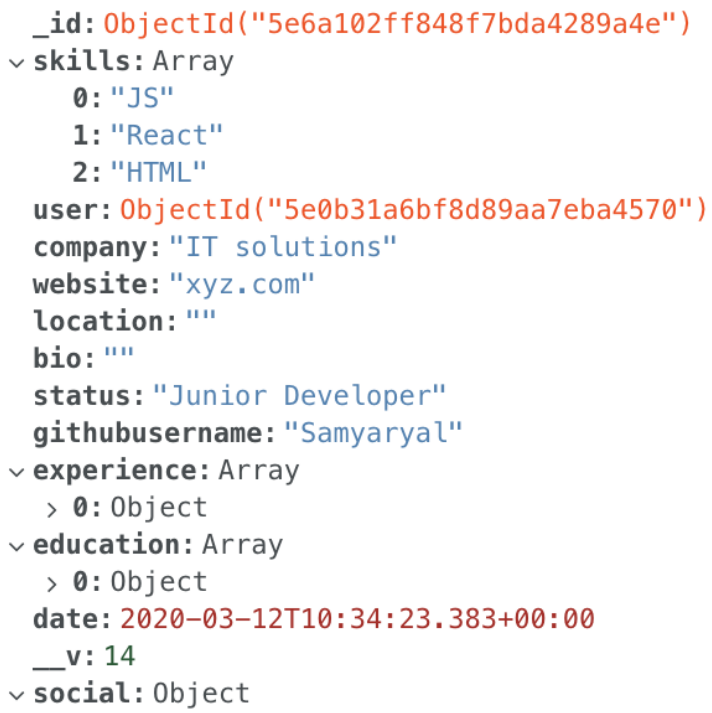
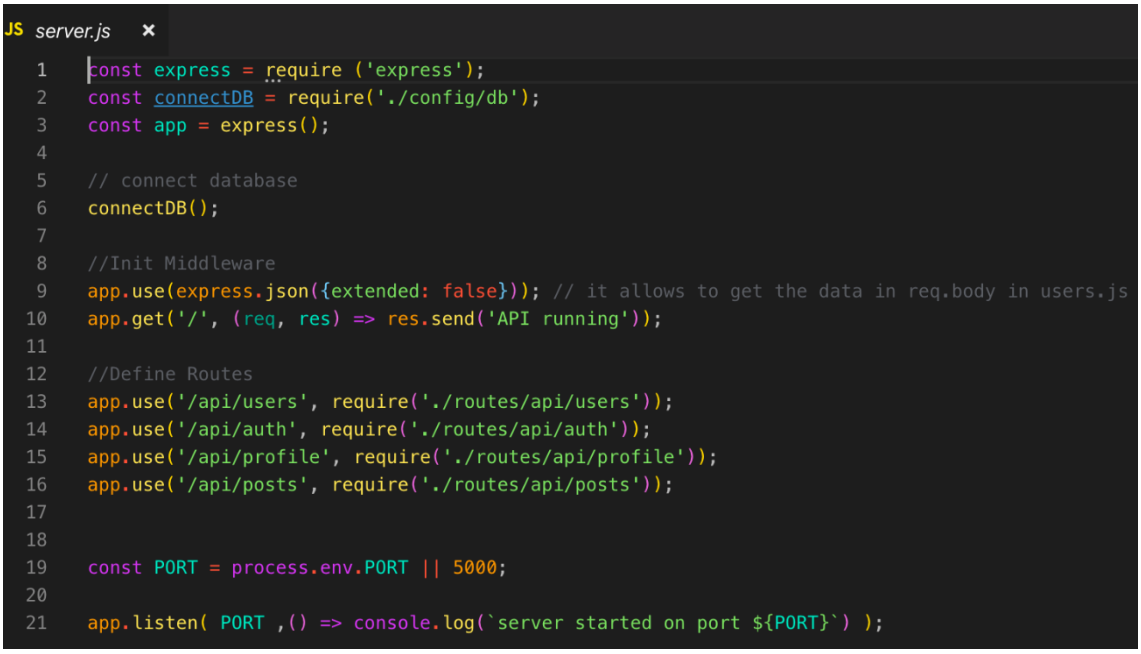


Figure 17. Screenshot of User Profile in JSON format from the database.

4.3 Back-end Logic

After successful registration in MongoDB, mongoose package is used to make a connection to Mongo URI. Combining with express.js. the following code is a main entry file of server:



```
JS server.js ×

1   const express = require ('express');
2   const connectDB = require('./config/db');
3   const app = express();
4
5   // connect database
6   connectDB();
7
8   //Init Middleware
9   app.use(express.json({extended: false})); // it allows to get the data in req.body in users.js
10  app.get('/', (req, res) => res.send('API running'));
11
12  //Define Routes
13  app.use('/api/users', require('./routes/api/users'));
14  app.use('/api/auth', require('./routes/api/auth'));
15  app.use('/api/profile', require('./routes/api/profile'));
16  app.use('/api/posts', require('./routes/api/posts'));
17
18
19  const PORT = process.env.PORT || 5000;
20
21  app.listen( PORT ,() => console.log(`server started on port ${PORT}`) );
```

Figure 18. Server implementation with Node.js and mongoose.

Stored in .env file as an environment variable, MongoDB URI is connected by mongoose and permits us to all the collections in the database. All the endpoints are accessed in server.js and all those endpoints are happening in front-end of the application.

4.3.1 Routes

The 'routes' folder consists of all the server-side logic of the application. Some of the routes were made private meaning that the user has to login to the application so view those routes. The REST APIs, as well as the routing, are being implemented in this section. The 'auth.js' and 'users.js' file contains the logic and the route for all the user-related activities like register and login page, saving the registered user to the database and checking the information whenever the same user tries to log in to the application. Similarly, 'profile.js' and 'posts.js' have different logic and routes handling the user's profile and post element which is explained more in detail later.

Table 1. List of all the routes created for an authentication process.

Route	Purpose
/api/users	POST save (register) a new user: name, email, password in the database.
/api/auth	POST authenticate a user and get token
/api/auth	GET test the routes for authenticated users

In the dashboard, an app must allow users to sign up and sign in.

Profile.js file in the routes folder is responsible for all the logic for the user's profile like adding education, editing profile and other functions. The routes created in profile.js is presented below.

Table 2. List of all routes for users' profile.

Route	Purpose
/api/profile/me	GET gets the logged-in users' profile
/api/profile	POST create/update a profile
/api/profile	GET gets all the user's profile
/api/profile/user/:user_id	GET gets profile by user-id
/api/profile	DELETE users, profiles.
/api/profile/experience	PUT adds profile experience
/api/profile/education/	PUT adds profile education
/api/profile/experience/:exp_id	DELETE profile experience
/api/profile/education/:edu_id	DELETE profile education.
/api/profile/github/:username	GET users repository from GitHub.

In the above table, the REST APIs were created accordingly to get the expected result. Each of the APIs was tested using Postman in the initial phase and troubleshooting was done to get the expected result. The process of testing of API will be explained in next chapter. GitHub API was the only fetched API to get the information of the user's recent five git repositories. The picture below is the code to show the fetched API of GitHub.

```
router.get('/github/:username', (req, res) => {
    try {
      const options = {
        uri: encodeURI(`https://api.github.com/users/${
          req.params.username
        }/repos?per_page=5&sort=created:asc&client_id=${config.get(
          'githubClientId'
        )}&client_secret=${config.get('githubSecret')}`),
        method: 'GET',
        headers: { 'user-agent': 'node.js' }
      };

      request(options, (error, response, body) => {
        if (error) console.error(error);

        if (response.statusCode !== 200) {
          return res.status(404).json({ msg: 'No Github profile found' });
        }

        res.json(JSON.parse(body));
      });
    } catch (err) {
      console.error(err.message);
      res.status(500).send('Server Error');
    }
  });
```

Figure 19. Fetching GitHub API to get user's recent five git repositories.

4.3.2 Testing APIs with Postman

Postman is a client tool for performing integration testing of API. API testing involves the collection of APIs and checking if they meet expectations for functionality, reliability, performance and security and returns the correct response (10 API Testing Tips for Beginners (SOAP & REST) | Complete Guide, 2020). API testing helps to determine if the output is well structured and useful to another application. It also checks the response based on the request parameter and calculates the amount of time the API takes to retrieve and authorize the data. It also has a feature that enables to test the same request over different environments with different specific variables.

All the API's in the application were tested in the postman by sending a request to the webserver and getting the response back. It works in the server-side and makes sure that each API endpoint is working as expected. In the postman, the user has to create a request and postman look at the response to check the element that the user is looking

for. Postman's test is against the JavaScript code that runs after a request is sent and the response has been received from the server.

Postman provides a collection of API calls, and one has to follow the collection of API calls for testing APIs of application. An API response consists of the body information, headers, and the status code. A request like Add, Delete and update can send parameters, authorization details, or any data that are required. When the request is made, postman displays the response received from the API server in a way that lets response to examine, visualize and if necessary, troubleshoot (Responses, 2020). User can save all the responses of a request and can be reloaded whenever is necessary. The postman body tabs consist of several tools to help understand the response quickly. The pretty model formats JSON and XML responses so they are easier to view, the raw view is a large text area with a response body, and it can be minified. Similarly, the preview tab renders the responses in a sandboxed iframe. Headers are displayed as key-value pairs under the headers tab with the description of the header according to the HTTP specification. Also, postman breaks down the size of the responses into the body and headers which are approximate (Requests, 2020).

Different request methods are available in the Postman. Below are the main four request methods frequently used in the application.

POST Request => For Creating or Updating data,

PUT Request => For Updating data

GET Request => For retrieving/Fetching data and

DELETE Request => For Deleting data

4.4 Front-end Logic

The majority of UI and logic related components responsible for various functions are kept in the "components" folder. ReactJS heavily rely on component-driven development; hence everything is componentized, such as the dashboard, landing page, Navbar, login, register and profile. The component-driven approach allows moving components around with ease and speeds up the development.

As described previously in this thesis, React hooks are the new way of developing ReactJS application, allowing class components to be abandoned and clean functional components to be utilized instead. React components with rich UI designs are lengthy, only reducers and actions and some screenshots of the app on a browser will be shown to demonstrate the logic and results.

Register

```
//Register USer
export const register =({ firstname, lastname, email, password}) => async dispatch => {
    const config = { headers: {'Content-Type': 'application/json'} };
    const body = JSON.stringify({firstname, lastname, email, password});
    try {
        const res = await axios.post ('/api/users', body, config);
        dispatch ({ type: REGISTER_SUCCESS, payload: res.data});
        dispatch(loadUser());
    } catch (err) {
        const errors = err.response.data.errors;
        if(errors){ errors.forEach(error => dispatch(setAlert(error.msg, 'danger')));}
        dispatch({ type: REGISTER_FAIL});
    }
};
```

Figure 20. Action creators for signing up.

```
import { REGISTER_SUCCESS,  USER_LOADED, LOGIN_SUCCESS, LOGOUT, ACCOUNT_DELETED } from '../actions/types';
const initialState = {
    token: localStorage.getItem('token'),
    isAuthenticated: null,
    loading: true,
    user: null
}
export default function(state = initialState, action){
    const {type, payload} = action;
    switch(type){
        case USER_LOADED:
            return {  ...state, isAuthenticated: true, user: payload }
        case REGISTER_SUCCESS:
        case LOGIN_SUCCESS:
            localStorage.setItem('token', payload.token);
            return{ ...state,...payload,isAuthenticated: true }
        case LOGOUT:
        case ACCOUNT_DELETED:
            localStorage.removeItem('token');
            return{...state, token: null, isAuthenticated: false, user: null}
        default: return state;
    }
}
```

Figure 21. Reducer for signing up.

When user requests for signing up, providing all form's validations passed register function is called. The user loaded action creators are dispatched immediately. This

passed a new state to the reducers and implies that the app is doing some asynchronous work which requires waiting time.

During this time, a POST request to /api/users is made. It contains the data from the sign-up form that the user just submitted. The alert message pops up in the screen if any of the value on the form is missing. The function is wrapped in a try-catch block, which throws any errors in red text that may happen during the request. If there are no errors register success is dispatched. This stopped the fetching process and then redirect the user to the dashboard where create profile massage is displayed and following the link user is then redirected to the create-profile page. Else, when an error occurs, register fail is dispatched and a message appears on the screen.

Sign In

The sign-in process also follows the same pattern as sign up.

```
//Login USer
export const login =(email, password) => async dispatch => {
    const config = {
        headers: {'Content-Type': 'application/json'}
    }
    const body = JSON.stringify({ email, password});
    try {
        const res = await axios.post ('/api/auth', body, config);
        dispatch ({ type: LOGIN_SUCCESS, payload: res.data});
        dispatch(loadUser());
    } catch (err) {
        const errors = err.response.data.errors;
        if(errors){
            errors.forEach(error => dispatch(setAlert(error.msg, 'danger')));
        }
        dispatch({type: LOGIN_FAIL});
    }
};
```

Figure 22. Action creator for signing in.

All the permission to access the authorized routes are done from here. After a successful sign in using email and password, JWT is set to the local browser. If the login is failed a message is thrown.

Sign Out

```
//logout /clear profile
export const logout = () => dispatch => {
    dispatch ({type: CLEAR_PROFILE});
    dispatch ({type: LOGOUT});
};
```

Figure 23. Action creator for signing out.

Sign out is the reverse version of sign in. JWT is removed from the local storage. User is then redirected to the main page to let the user knows that the user is logged out successfully from the app.

5 RESULTS AND DISCUSSION

The prototype application for demonstrating the use of Full Stack JavaScript was developed successfully. NodeJS with ExpressJS and several other tools were used for developing the backend while ReactJS was used for developing the front end. A NoSQL database, MongoDB along with Mongoose was used for storing the data into the database cloud.

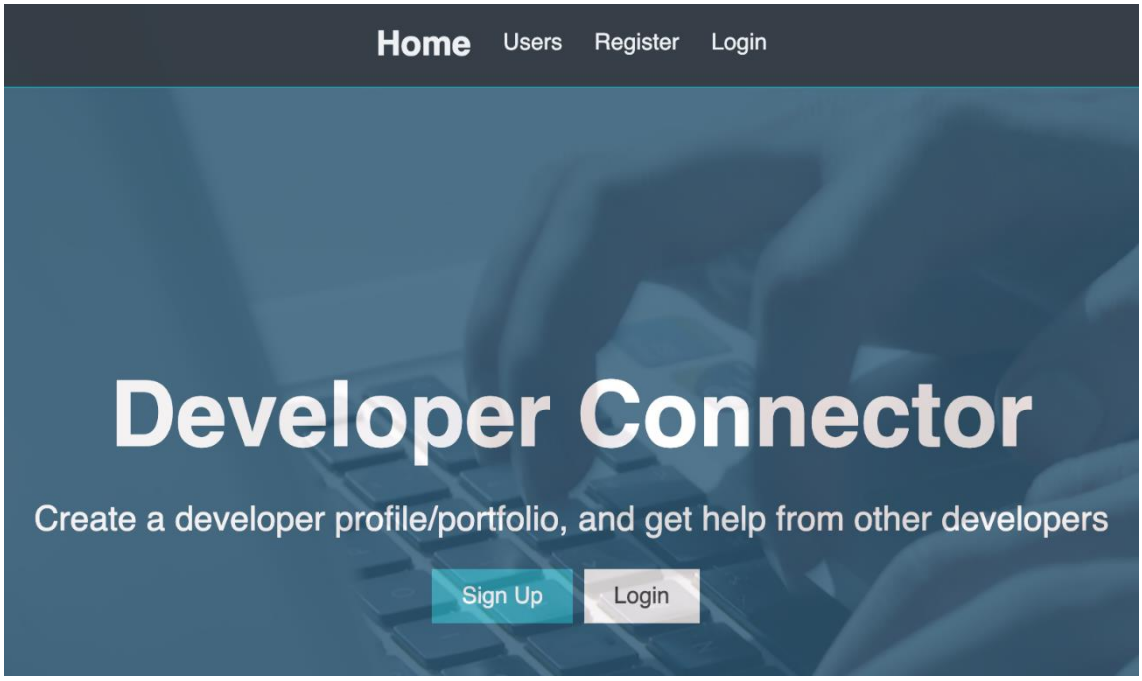


Figure 24. The landing page of the application.

A simple landing page of the application is shown in Figure (24). It has a different navigation bar to navigate different pages of the application like login, register and developers bar. However, some pages of the application can be accessed only by logging in to the application. The home page allows users to see the profile of other users who have already created a profile in the application. All the components in the application were designed in-order them to fit in the size of the end-user screen. All the

contents are mostly rendered conditionally from a single main component App.js, in the application. Different pages are redirected after the authentication.

A view of the login system of the project is presented in Figure 25.

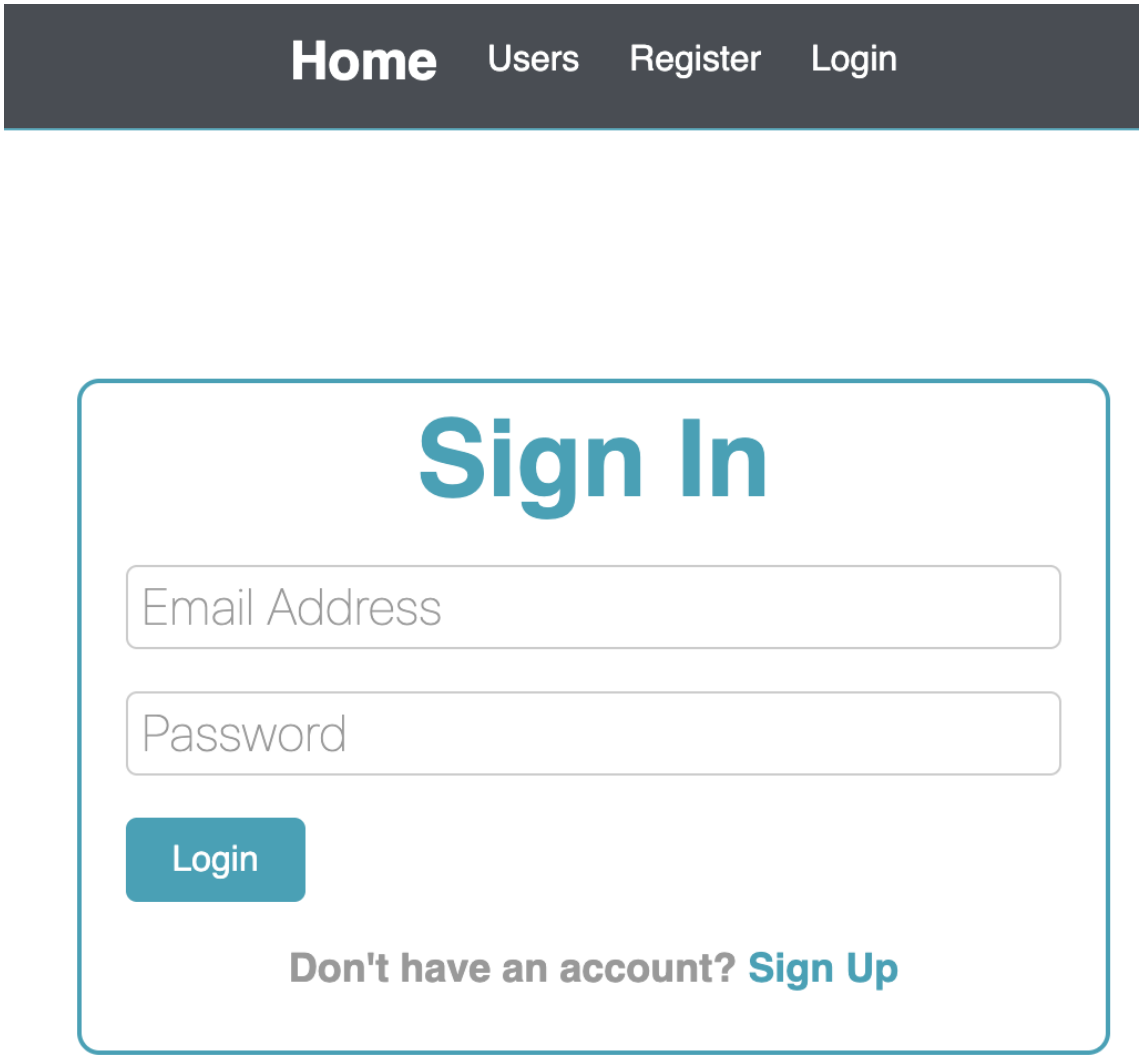


Figure 25. Login view of an application.

A view of the login system is displayed in Figure 25. More features like Add-education, Add-experience and Delete account can be found in the dashboard of a user. A view of user dashboard is presented below.

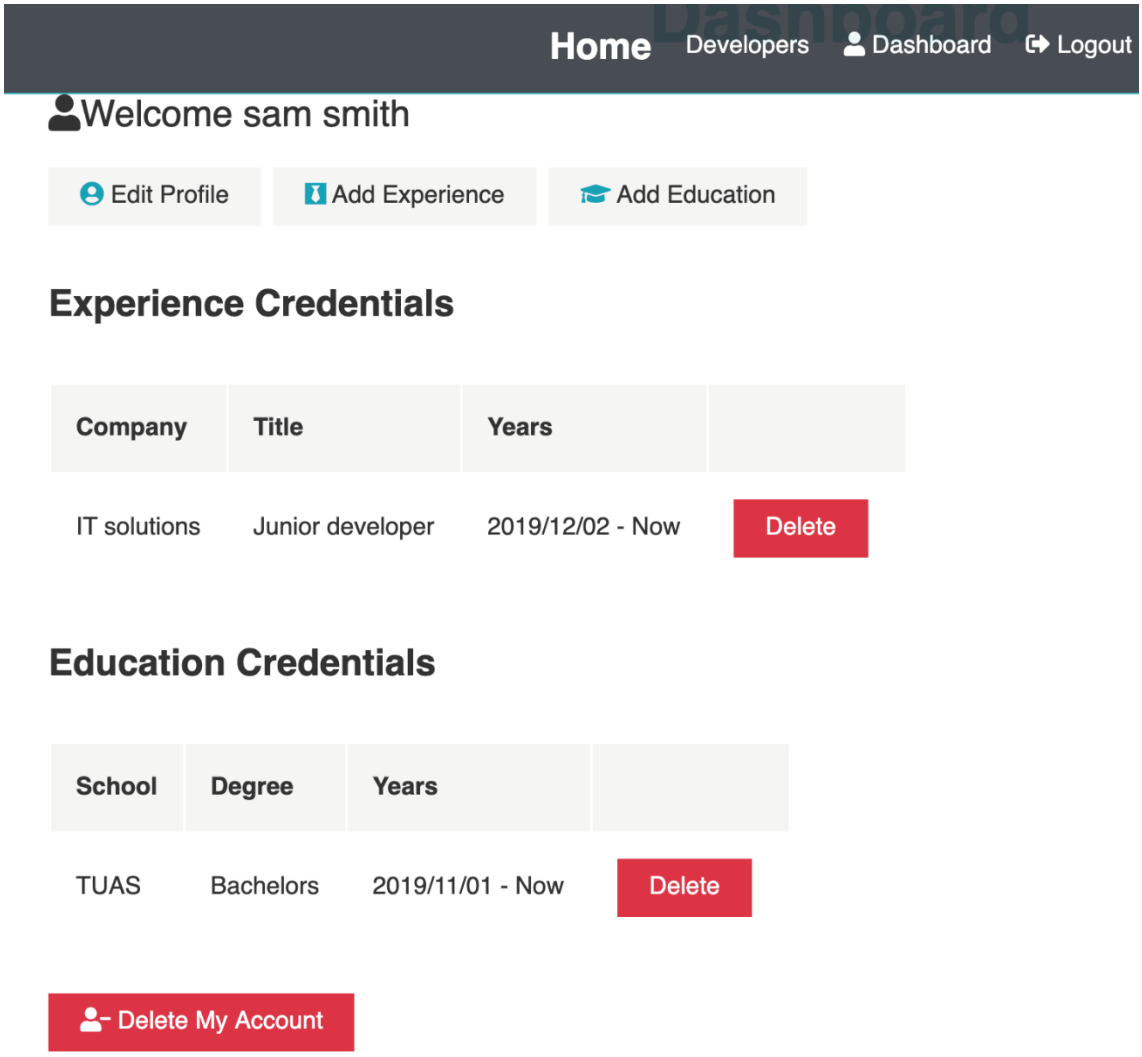


Figure 26. A view of user dashboard with different features.

As seen in Figure 26 user can update the profile and logout from the system or delete the account permanently.

The MERN Stack proved to be one of the best stacks to develop a full-stack application. The learning and developing of the application were a great experience using the most popular stack. The end product was a site for creating a portfolio. Users were able to register in the app's system and, were able to create a portfolio by filling the forms accordingly. Users were also able to edit and update their profile. App has a public route where user can view the profile of all the users who have created a portfolio in the system. Overall, this application fulfilled CRUD operations. It explains on creating routes

and APIs using Express on NodeJS framework. Briefly defines the attachment of MongoDB and its benefits. Moreover, MongoDB uses JSON format to store data which is easy an easy way to transfer data and work on it. Apart from this, NodeJS, ExpressJS and MongoDB being a popular technology, it has a huge community in case of any difficulties.

The prototype built is stable but must be treated with care if deployed in production since there are still some areas for improvement. For example, testing of the application is still missing. Selecting the appropriate tools for the testing must be done. Besides, new features could be implemented such as setting a profile image. If properly nurtured, this application could be a success and widely adopted by multiple companies in Finland.

6 CONCLUSION

This thesis aimed to study and build a full-stack web application that utilizes the power of JavaScript under MERN stack and develop a prototype based on it. Each technology required a significant amount of time to study in detail and to put the idea into implementation. The core concepts and main benefits of using the stack are heavily discussed followed by the in-depth explanation of the implementation process.

The prototype built was an application for creating a portfolio. Users were able to register in the web app by creating an account and logging in with their credentials into the system. There were protected routes so only the registered users could create a portfolio by filling in the form and were able to perform Create, Read, Update and Delete (CRUD) operations. A normal user who is not logged in can access the profile of other users who created the profile in the system. However, that user is restricted to use other features of the application like creating a profile directly without registering and performing CRUD operations. The frontend of the application was done using React and Redux. The latest feature of React which is React Hooks was also implemented and the application was made responsive using a media query.

Last but not the least, the thesis is the documentation of key concepts of MERN stack and its implementation for full-stack web development, it does not explain the suitability of MERN as the best solution for developing large enterprise solutions. NodeJS is unable to tackle tasks that require heavy data processing and computational power but having said so, Node has evolved in short space of time and with the backup of good community, it will only continue to gain more popularity.

REFERENCES

Bhardwaj, P., 2018. *Analysis Of Stack Technology: A Case Study Of MEAN Vs. MERN Stack*. [online] Ijircce.com. Available at: <http://www.ijircce.com/upload/2018/april/66_Analysis.pdf> [Accessed 24 April 2020].

Blischak, J., Davenport, E. and Wilson, G., 2016. A Quick Introduction to Version Control with Git and GitHub. *PLOS Computational Biology*, 12(1), p.e1004668.

Chettri, N., 2016. *A Comparative Analysis Of Node.Js (Server-Side Javascript)*. [online] Repository.stcloudstate.edu. Available at: <https://repository.stcloudstate.edu/csit_etds> [Accessed 1 May 2020].

Code, V., 2020. *Documentation For Visual Studio Code*. [online] Code.visualstudio.com. Available at: <https://code.visualstudio.com/docs> [Accessed 14 April 2020].

Db-engines.com. 2020. *Mongodb System Properties*. [online] Available at: <https://db-engines.com/en/system/MongoDB> [Accessed 7 February 2020].

Docs.npmjs.com. 2020. *About Npm | Npm Documentation*. [online] Available at: <https://docs.npmjs.com/about-npm/> [Accessed 21 February 2020].

Elboim, N., 2017. *How To Greatly Improve Your React App Performance*. [online] Medium. Available at: <https://medium.com> [Accessed 7 May 2020].

freeCodeCamp.org. 2020. *What Exactly Is Node.Js?*. [online] Available at: <https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5> [Accessed 6 May 2020].

Fullstackopen.com. 2020. *Fullstack Part3 |*. [online] Available at: <https://fullstackopen.com/en/part3/node_js_and_express> [Accessed 7 May 2020].

Haviv, A., 2014. *MEAN Web Development*. Birmingham, UK: Packt Pub.

Jacksi, K. and Abass, S., 2019. *Development History Of The World Wide Web*. [online] International Journal of Scientific & Technology Research. Available at: <https://www.researchgate.net/publication/336073851_Development_History_Of_The_World_Wide_Web> [Accessed 24 April 2020].

Jwt.io. 2020. *JWT.IO - JSON Web Tokens Introduction*. [online] Available at: <https://jwt.io/introduction/> [Accessed 2 March 2020].

Katalon Solution. 2020. *10 API Testing Tips For Beginners (SOAP & REST) | Complete Guide*. [online] Available at: <https://www.katalon.com/resources-center/blog/api-testing-tips/> [Accessed 15 April 2020].

MDN Web Docs. 2020. *Express/Node Introduction*. [online] Available at: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction> [Accessed 7 May 2020].

MongoDB. 2020. *The Modern Application Stack – Part 1: Introducing The MEAN Stack | Mongodb Blog*. [online] Available at: <https://www.mongodb.com/blog/post/the-modern-application-stack-part-1-introducing-the-mean-stack> [Accessed 14 February 2020].

Mongoosejs.com. 2020. *Mongoose ODM V5.9.12*. [online] Available at: <https://mongoosejs.com> [Accessed 20 March 2020].

npm. 2020. *Bcryptjs*. [online] Available at: <https://www.npmjs.com/package/bcryptjs> [Accessed 15 April 2020]

npm. 2020. *Concurrently*. [online] Available at: <https://www.npmjs.com/package/concurrently> [Accessed 15 April 2020].

npm. 2020. *Express-Validator*. [online] Available at: <https://www.npmjs.com/package/express-validator> [Accessed 13 April 2020].

Postman Learning Center. 2020. *Requests*. [online] Available at: <https://learning.postman.com/docs/postman/sending-api-requests/requests/> [Accessed 3 March 2020].

Postman Learning Center. 2020. *Responses*. [online] Available at: <https://learning.postman.com/docs/postman/sending-api-requests/responses/> [Accessed 5 March 2020].

Reactjs.org. 2020. *Introducing Hooks – React*. [online] Available at: <https://reactjs.org/docs/hooks-intro.html> [Accessed 8 February 2020].

Reactjs.org. 2020. *React – A Javascript Library For Building User Interfaces*. [online] Available at: <https://reactjs.org/> [Accessed 28 February 2020].

Redux.js.org. 2020. *Redux*. [online] Available at: <https://redux.js.org/introduction/core-concepts/> [Accessed 1 March 2020].

Redux.js.org. 2020. *Redux*. [online] Available at: <https://redux.js.org/introduction/three-principles> [Accessed 2 March 2020].

Stack Overflow. 2020. *Stack Overflow Developer Survey 2019*. [online] Available at: <https://insights.stackoverflow.com/survey/2019> [Accessed 27 April 2020].

Anderson, D., 2014. *HOW NODE.JS CAN ACCELERATE ENTERPRISE APPLICATION DEVELOPMENT*. [online] Dsimg.ubm-us.net. Available at: <https://dsimg.ubm-us.net/envelope/346183/287412/1415125585_modulus_nodejs.pdf> [Accessed 2 May 2020].

Martonca, E., 2015. *Why All The Hype About Node.Js?*. [online] Webcitation.org. Available at: <https://www.webcitation.org/6ePrX7R2R> [Accessed 3 May 2020].