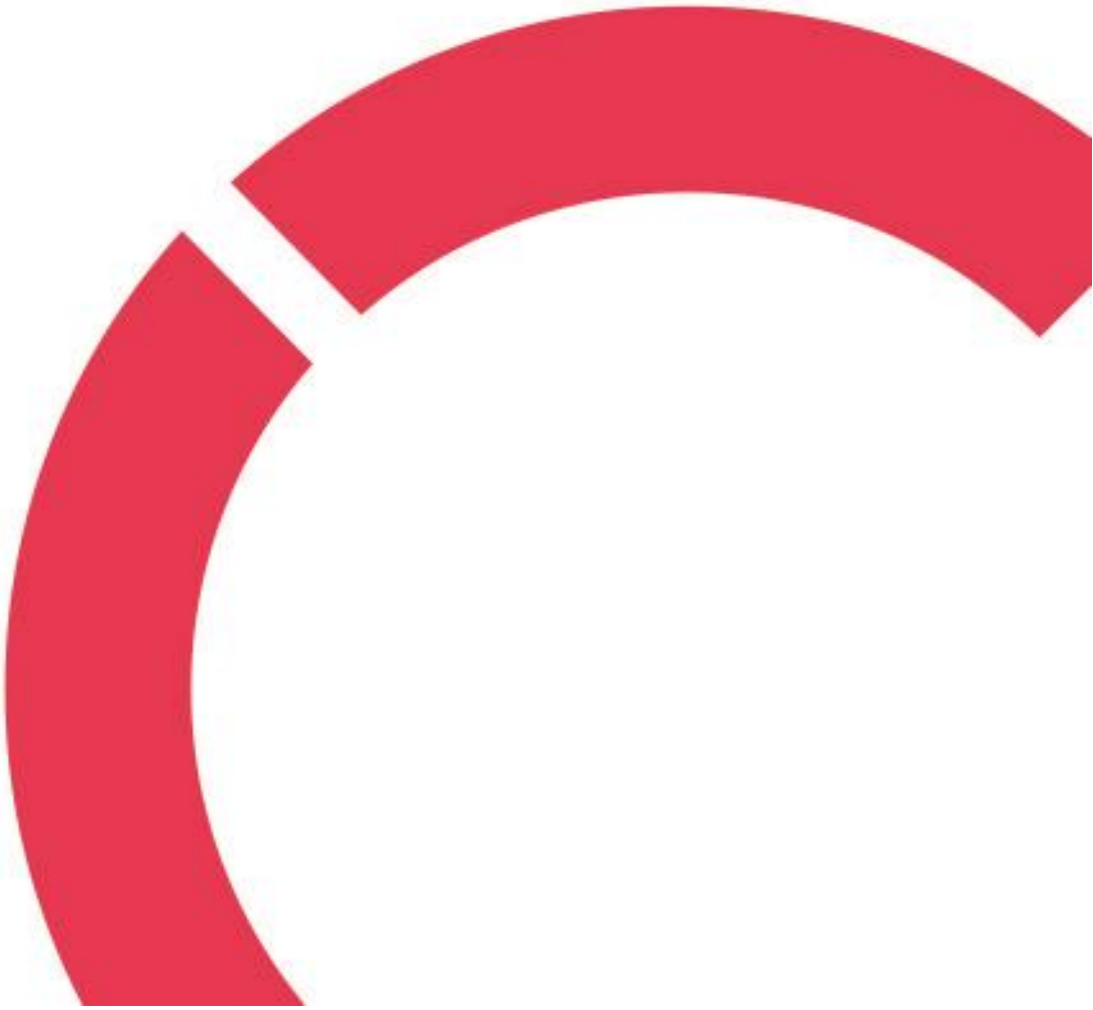
Hritik Khadka

MERN STACK BLOG APPLICATION

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Bachelors in engineering, Information Technology
June 2023



ABSTRACT

Centria University of Applied Sciences	**Date** 14/ 06 / 2023	**Author** Hritik Khadka
Degree programme Bachelor's in engineering, Information Technology		
Name of thesis MERN STACK BLOG APPLICATION		
Centria supervisor Heikki Ahonen		**Pages** 31+3

Over the years, web development has undergone significant changes in both frontend and backend development. With the emergence of various web technologies and stacks, developers have had to adapt and choose the best tools and platforms to build web applications. One of the most popular stacks for full stack web application development is the MERN stack.

This report aims to provide an in-depth analysis of the MERN stack, its components, its importance in modern days, and a practical demonstration by building a simple blog application. The MERN consists of four technologies, Mongo DB, Express JS, React JS, and Node JS, each of which plays a critical role in the full stack web development process.

Key words
Express JS, Full Stack, MERN, MongoDB, NodeJS, React JS, Web Development

LIST OF ABBREVIATIONS

LAMP	Technology stack: Linux, Apache, MySQL, and PHP
SPA	Single Page Application
MEAN	Technology stack: MongoDB, Express JS, Angular JS, and Node JS
MERN	Technology stack: MongoDB, NodeJS, Express JS, and React JS
DBMS	Database Management System
API	Application Programming Interface
UI	User Interface
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
XHR	XMLHttpRequest
ECMA	European Computer Manufacturers Association
NoSQL	Non-Structured Query Language
ORM	Object Relational Mapping
EJS	Embedded JavaScript
JSX	JavaScript XML
NPM	Node Package Manager
CLI	Command Line Interface
JWT	Json Web token

ABSTRACT
LIST OF ABBREVIATIONS
CONTENTS

PICTURES

CODES

TABLES

1 INTRODUCTION

Over the past years, the development of web applications has experienced a notable transformation because of technological advancements and evolving user demands (Subramanian 2019, 1). Previously, building web application primarily relied on LAMP stack. This stack involved utilizing Linux as the operating system for the server, Apache as the web server, MySQL as the DBMS, and PHP as the server-side language. (Serdar 2018; Subramanian 2019, 1; Vassallo & Garg 2016.) As web development progressed, a new approach called SPA emerged. SPA is a web application concept that eliminates the need to retrieve web page contents from the server to display new content. This innovation offered a smoother and more efficient browsing experience. In the early days, MEAN stack was changed to MERN since the popularity of front-end technology created by Facebook was gaining popularity and thus, replaced the "A" in MEAN with "R" which refers to React. (Subramanian 2019, 1.)

MERN stack provides simple and fast development for developing web applications. It helps in building and maintaining powerful and scalable web applications. MERN stack is one of the most famous choices for developing a complete and complex web application project because it allows creating a seamless and consistent user experience throughout the entire application. (Subramanian 2019, 1.)

This report will study about MERN stack, its components and how these components are used to make a web application. The main aim of this report is to introduce all the theoretical concepts of MERN stack, discuss why it is crucial for creation of modern web applications, and present a simple blog application.

2 FULL STACK WEB DEVELOPMENT WITH JAVASCRIPT

Full stack web development involves the client-side and server-side, where client-side is the frontend of the application and server-side is the backend (Vassallo & Garg 2016). This includes everything from database management and server-side scripting to APIs and other server-side technologies. The front end of a web application is everything that user interacts with. This normally includes designing and implementing the layouts of the application using HTML for defining structure, CSS for styling, and JavaScript for adding dynamic behaviour to the application. On the other hand, back end of a web application is responsible for handling all the server-side logics and functionalities. (Sharma 2022, chapter "Full-Stack Development Using MongoDB".)

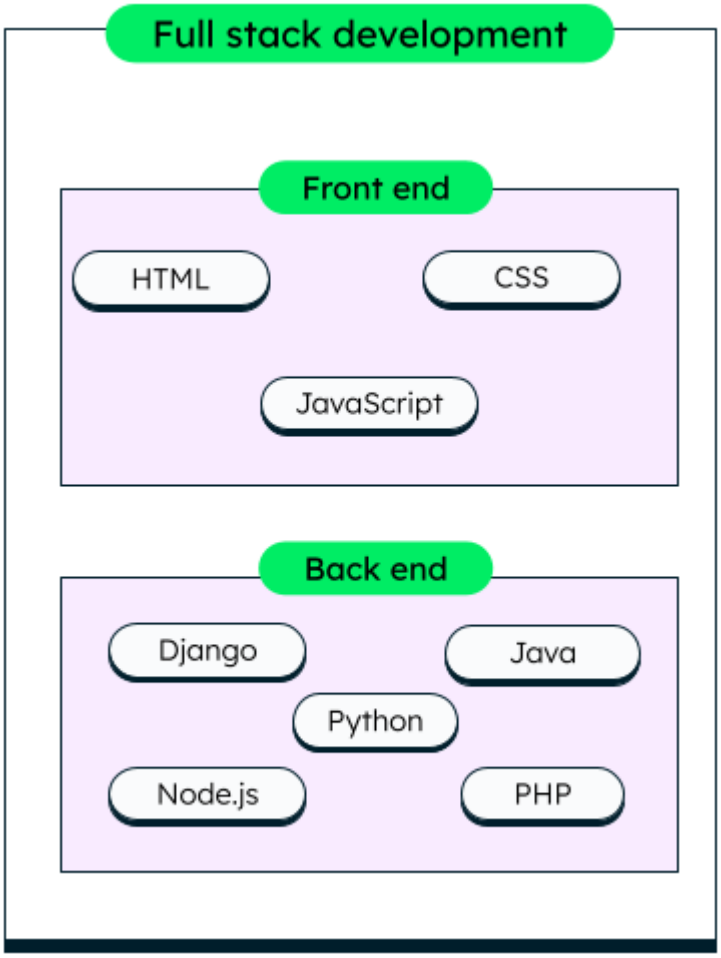


Figure 1: Components of a full stack development (MongoDB 2023)

The above figure 1 illustrates the components of a full stack development along with the technologies.

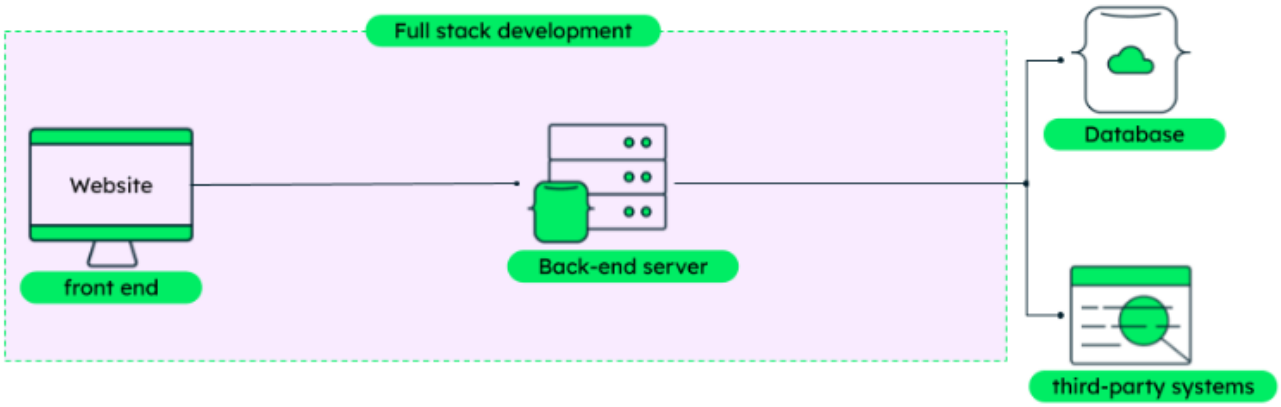


Figure 2: A Full Stack Development Workflow (MongoDB 2023)

The above figure 2 represents the demonstration of full stack development in an end-to-end workflow.

JavaScript, originally created at Netscape, is used for manipulating web pages with the help of API known as the DOM or Document Object Model. DOM is essentially a tree structure of JavaScript objects that is made available through a global object called “window”. This tree structure represents the HTML structure of the web page, starting with “window.document” for the html tag. Manipulating of pages can be performed by modifying this tree structure such as adding a new node as a child of an existing one to create new content. In the early days of the web, Microsoft introduced Jscript which had additional features including XHR which allowed developers to refresh web pages by making requests to external servers. (Northwood 2018, 159.) Then, a new standard called ECMAScript was released that introduced several features such as modules, arrow functions, let and const, default, classes, rest, and spread to functions parameters. ECMAScript is also known as ES6. (Haviv 2016, 4 .)

JavaScript is a lightweight, interpreted, and scripting language which was only used and run by browsers to interact with the User Interface of a website. However, with the introduction of Node JS by V8 engine, JavaScript can also be used to run on a server with a new standard library to support server-side requirements. This means that developers can write JavaScript code that runs on the client side and on the server-side. (Haviv 2016, 2-3.) Client-side JavaScript includes the frontend side of a web application, such as validating user input, manipulating UI, and handling user interactions. On the other hand, server-side JavaScript can help to connect and modify database with server-side code. Moreover, by having this ability to use JavaScript on both the client and server side, developers can build full stack web applications more efficiently and seamlessly. (Northwood 2018, 159-160.)

JavaScript is an incredibly versatile and flexible language, and it has become a fundamental component of modern web development. It enhances interactivity of websites by enabling various functionalities. JavaScript plays a key role in building dynamic and responsive web applications. (MDN Web Docs 1998-2023 "What is JavaScript".) All the four technologies of MERN are JavaScript-based (Hoque 2020, 13). JavaScript is used for both client-side and server-side code. So, it is crucial to know and be familiar with JavaScript to start learning about MERN stack. (Subramanian 2019, 14.)

3 MERN STACK

MERN stack comprises of various technologies that are used to develop modern web applications. It is a full-stack JavaScript framework for building effective and dynamic applications. MERN combines four cutting-edge technologies from the front-end to the back-end development. Node and Express bind the web backend together, MongoDB serves as the database, and react makes the frontend that user views or interacts with. (Hoque 2020, 13.) Figure 3 below shows the logos of each technology used in MERN stack.

Figure 3. Illustration of MERN stack logo (Java T Point 2011-2021)

3.1 MongoDB

The first technology in MERN stack is Mongo DB. It is a document-oriented database that is typically classified as part of NoSQL database category. It is a popular choice as a database because it lowers complexity compared to a relational database. (Luukkainen 2023, part 3 “Saving data to MongoDB”.) A document-based database means that it stores data in JSON-line document (Hoque 2020, 15) . Since Mongo DB is a document-oriented database, document is the unit of storage in it where many documents are stored in collections. There is presence of a unique identifier in every document in a collection used in the process of accessing data. Mongo DB helps developers to avoid translation layers, ORM, which means to map or convert object to relational tables. Mongo DB documents are like JSON objects which makes it easier to think of them as JavaScript objects. (Subramanian 2019, 10 ; Serdar 2018.)

As Mongo DB is schema-less, there is no problem in database while adding fields or columns in the application code (Serdar 2018). Also, Mongo DB is based on JavaScript. The query language in Mongo

DB is based on JSON, a lightweight data interchange format. In Mongo DB, programmers can perform specific operations such as create, search, delete documents by specifying operations in a JSON object. In MongoDB, data is not only stored in JSON format but also interchanged in the same format. This makes MongoDB popular choice for modern web applications that uses JavaScript-based technologies. With the use of JSON in MongoDB, storing, retrieving, and manipulating data in a document-based database management system becomes efficient. (Subramanian 2019, 10-11 ; MongoDB 2023.)

3.2 Express

The second technology used in MERN stack is Express. It is a server-side framework built in Node JS which allow to easily create a web server. It allows to handle API routing on the server side and HTTP methods such as GET, POST, and PUT. (Singh & Nirgudkar 2017, 3238 ; Hoque 2020, 14). Routing in Express means the process of determining application response in finding out an endpoint requested by a user. The Express parses the URL, headers, and parameters for developers which generates functionality required by web applications as a response. Moreover, programmers can also write middleware code in Express JS which is mainly used for general functionalities such as logging in and authentications. Data storage system is needed for any web application. Express provides flexibility to choose any database option such as MongoDB (Hoque 2020, 15). Also, template engines such as EJS are built in Express that helps to add dynamic data to the HTML and display the HTML page. (Subramanian 2019, 9.)

To summarize, Express is a web server framework designed for use with Node JS. It is like many other web server frameworks in terms of the functionalities it provides. With Express, you can create web servers that handle HTTP requests and responses, as well as implement various features such as routing, middleware, and templating. (Subramanian 2019, 9.)

3.3 React

The third MERN stack technology is React. It defines all the components of the MERN stack. It is a declarative and component-based JavaScript library that makes easy to create and maintain complex user interfaces. (Hoque 2020,). React helps to build user interfaces using components. All these components are put together to make a main component which displays the complete view of a web page. The major reason behind building multiple components for a web application is that it makes writing code

easier and helps developers focus on one part of the application at a time. (Subramanian 2019, 5-7; Meta Open Source 2023, “React”.)

3.3.1 Component

React is based on components, which are the basic building blocks for creating UI. Building components is the primary task in React, and they can be combined to create larger components that represent entire pages or views.(Meta Open Source 2023, “Your First Component”.) Components in a React application communicate with each other by sharing state information through read-only properties to child components and by callbacks to parent components. Overall, the concept of breaking an application into smaller components is to have logic separately and to write and understand the application in an easy way. (Subramanian 2019, 6.)

```
export default function MyApp() {
  return (
    <div>
      <h1>Welcome to my app</h1>
      <MyButton />
    </div>
  );
}
```

```
function MyButton() {
  return (
    <button>I'm a button</button>
  );
}
```

Code 1: Example of how to use Component (Meta Open Source 2023, Learn your first component)

3.3.2 JSX

While JSX looks like HTML, there are some differences (Matti 2023, “Introduction to React”). React components return JSX, which is ultimately compiled into JavaScript. The returned JSX is typically handled by Babel. (Subramanian 2019, 20.)

There are several rules to follow while writing JSX syntax. To return multiple elements or components, it should be wrapped within a single parent tag. For example, using empty tag also known as Fragment to wrap. Second, all the tags in JSX must be explicitly closed. As an instance, image tag must be <img/>. And finally, when writing React components using JSX, “camelCase” for attribute names are used. This

means that instead of using hyphens to separate words in attribute names, capital letters are used to denote the beginning of a new word.(Meta Open Source 2023, “Writing Markup with JSX”).

3.3.3 Props

In the world of React, components rely on props as a means of communication between each other. Props is used to pass data to components (Luukkainen 2023, “Introduction to React”). A parent component can pass data and information down to its child components by provided those data with props. Props may bear some resemblance to HTML attributes, but the difference is that they can accommodate any JavaScript value from simple data tyles to more complex objects, arrays, and functions. Passing down the props is achieved by using curly brackets. Overall, the use of props in React promotes code reusability, making it effective for developing scalable and maintainable web applications. (Meta Open Source 2023, “Learn passing props to a component”.)

3.3.4 React Hooks

React provides built-in Hooks that start with the word *use*. For instance, “useState” which states a variable that can be updated directly. Some other examples of react hooks are “useEffect”, “useContext”, “useRef”, and “useMemo”. (Meta Open Source 2023, “Using Hooks”.) Below is an example of how “useState” hook is used in a React application.

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

Code: 2 Importing hooks (Meta Open Source 2023, “Using Hooks”)

In code 2, the built-in React JS hook “useState” is imported from react. Then, it is used in the following way.

```
function MyButton() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <button onClick={handleClick}>
      Clicked {count} times
    </button>
  );
}
```

Code 3: Example of how to use hook (Meta Open Source 2023, “Using Hooks”)

In the above code 3 example, after importing the use State hook from react, state variable is added inside the component. The current state is “count” and the function which allows to modify it is “setCount”.

3.3.5 React Router

React is primarily focused on the presentation of a web application and handling interactions within a single component. However, when it comes to managing transitions between different views of a component and keeping the browser URL in sync with the current state of the view, additional functionality is needed. This is where routing comes into play. React-Router provides this functionality and manages navigation in a React application allowing for seamless transition between different views without having to load the entire page from the server (Hoque 2020,101 ; Luukkainen 2023, part7 “React Router”). React-Router is an easy-to-use library that simplifies the process of navigation. (Subramanian 2019, 11.)

3.4 Node

Finally, the Node JS is the last technology used in MERN stack. Node JS is a JavaScript run-time that allows to execute JavaScript code outside of a browser. It is based on Google’s V8 JavaScript engine which is the same engine that runs in the Chrome web browser (Serdar 2018; Luukkainen 2023, part3 “Node.js and Express”.) Modules in Node JS are like libraries. Developers can use the “require” keyword for including the functionality of another JavaScript file. Thus, splitting code into various modules can be achieved for better code maintenance. The default package manager for Node JS is NPM which is used to install packages and dependencies (Hoque 2020, 13; Subramanian 2019, 7-9.)

Node's methodology enables it to manage a higher number of connections using less memory compared to various competing architectures such as Apache, Ruby on Rails, and Java. So, it has gained popularity for developing web servers, REST APIs and real-time applications. (Serdar 2018.)

3.5 Architectural Structure of MERN Stack

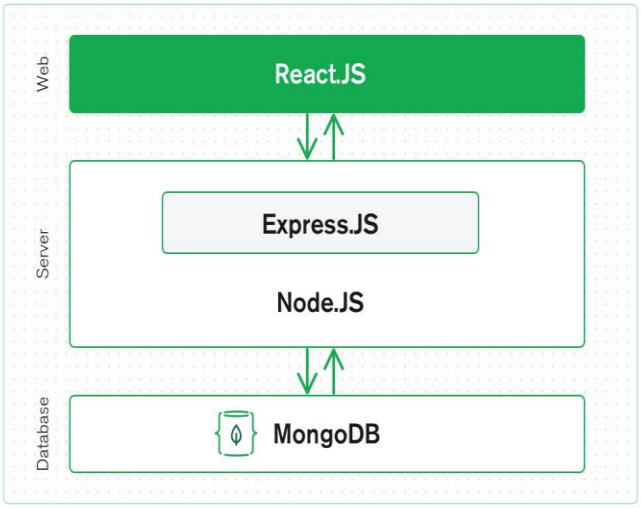


Figure 4. Architectural representation of MERN Stack (MongoDB 2023)

The 3-tier diagram above depicts the architectural structure of MERN stack. To elaborate more about what picture shows, first tier includes React JS used for client-side, Express and Node as server-side and Mongo DB for database.

4 PROJECT IMPLEMENTATION

The purpose of this thesis was to develop and analyse a MERN stack application that serves as a blog platform. The primary goal was to create a user-friendly and interactive web application where users can register, sign in, and logout. Once logged in, users can browse through a collection of posts made by various users. Each post can be clicked on to view its complete content, but editing capabilities are limited to the original author of the post. Non- authors can only view the posts, whereas authors can edit their own posts by clicking on the edit button, making some changes to the original post, and clicking on the update button at the bottom of the page. The application also allows users to create new posts, enabling them to express their thoughts and ideas. To enhance the text editing experience, the project incorporates the use of React Quill, an in-built user-friendly interface that enables users to edit the text using various formatting options such as bold, italic, underline, bullet points, numbering, and more. The users can also upload pictures as the content of their posts. Finally, users could logout from the application. Throughout the development and analysis of this MERN blog application, various aspects such as user experience, authentication, and frontend-backend integration will be explored and evaluated.

4.1 Setting up the Development Environment

Setting up an appropriate development environment is crucial when working on a project. Selecting the right tools and environment significantly impacts the efficiency and productivity of the development process of the project.

For this project, Visual Studio Code was used as a code editor that offers several features and boosts productivity and code maintenance. Visual Studio Code is a cost-free coding editor that facilitates convenient and efficient way to begin. It allows coding in multiple programming languages, eliminating the need to switch between different editors. It also offers extensive language support including Python, Java, C++, JavaScript, and various others. (Microsoft 2023). In addition, there were various third-party tools and packages that was used for the development process. These tools provided additional functionalities and made the development tasks easier. These tools and packages are described below in the installation and configurations section.

4.1.1 Installation and Configurations

In this section, installation and configuration of different packages and tools are carried out. NPM was used in this project to install different packages and manage their dependencies.

NPM is a powerful package manager and the world's largest software registry for JavaScript. There are three key components in NPM: the website, CLI, and the registry. The website serves as a platform for developers to discover packages and manage public and private packages. The CLI is the primary interface for developers to interact with npm. And the registry is a vast database the houses a wide range of JavaScript. Developers use registry to access and incorporate existing packages into their applications or use them to suit their specific needs. (NPM 2023.) The installation of Node.js, Express.js, React.js, MongoDB, Mongoose, Nodemon, Bcrypt.js, and Multer are described below.

To provide the runtime environment for executing server-side JavaScript code for this project, Node.js version 16.19.0 was installed from the official website of Node.js. After installing Node.js, the next step was to initialize the project using the command "npm init" in the project's root directory. This then creates a "package.json" file which allows developers to define other dependencies for various tasks such as building and running the application.

After installing the Node.js, the next step was to install Express.js for adding server-side logic in the application. The version of Express used in the application was 4.18.2. To include Express.js into the project, "npm install express" command was run in the project's root directory.

Then, next step was to install React.js for building the frontend of the blog application. The version of the React.js used at the time of developing application is 18.2.0. Before, "npx create-react-app" command was used to create a react app. With this method, the development experience was slow and less efficient. So, for this project, "npm create vite@latest" was used. Vite is a powerful development tool that makes development experience fast for modern web projects. (Vite n.d.) Additionally, React.js packages such as react router dom, react quill, and react icons were used for extra functionalities. React router dom was used for routing within the application. It provides components such as <BrowserRouter>, <Route>, <Routes>, <Link>, and more that are required for defining routes and rendering components. React quill was used as a text editor component and react icons was used for adding icons in the application.

Then, MongoDB was installed. There is flexibility to choose between installing MongoDB locally on computer or using cloud-based service MongoDB atlas. For this project, MongoDB atlas was used. After creating an account, a new cluster was created, and all the necessary settings were filled. Finally, a connection string was obtained which could be used to connect to the database from the application. The connection string contains the necessary information such as username and passwords to establish a connection to MongoDB Atlas cluster. The created cluster for this application can be seen in figure 5 below.

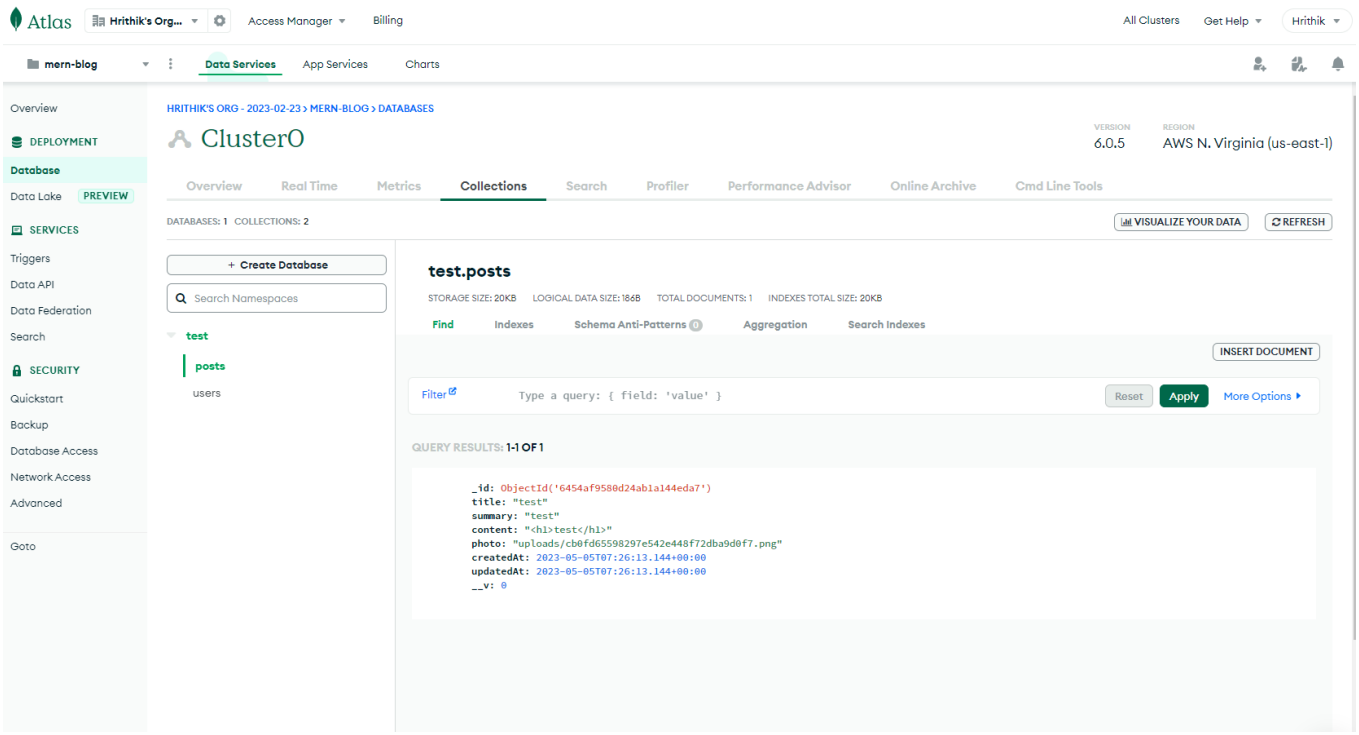


Figure 5: Screenshot of MongoDB Atlas

After that, to install Mongoose, command "npm install mongoose" was run in the backend project directory. The version of mongoose used in this project was 6.10.1. Mongoose was used to interact with MongoDB database by connecting with the connection string discussed before. The required models and schemas for the project were created and defined using mongoose that will be described in 4.2.2.

Nodemon is a tool that is used for automatically restarting server during development. To install nodemon, "npm install nodemon" command was run in the project's folder. After installing, scripts in the package.json was modified for running the server using "nodemon". (NPM 2023.)

Bcrypt JS is a JavaScript library used for hashing passwords and passwords comparison which is a crucial aspect of a secure authentication system. It can be installed using the command “npm install bcryptjs”. (NPM 2017.) In this project, the version of Bcrypt JS installed was 5.1.0.

Multer is a middleware used for uploading files. To install it, “npm install –save multer” command was run. (NPM 2022.) The version used at the time of development was 1.4.5-lts.1.

After successfully installing all the required tools and packages, the development process was carried out. The final files and folder structure can be seen in the image below.

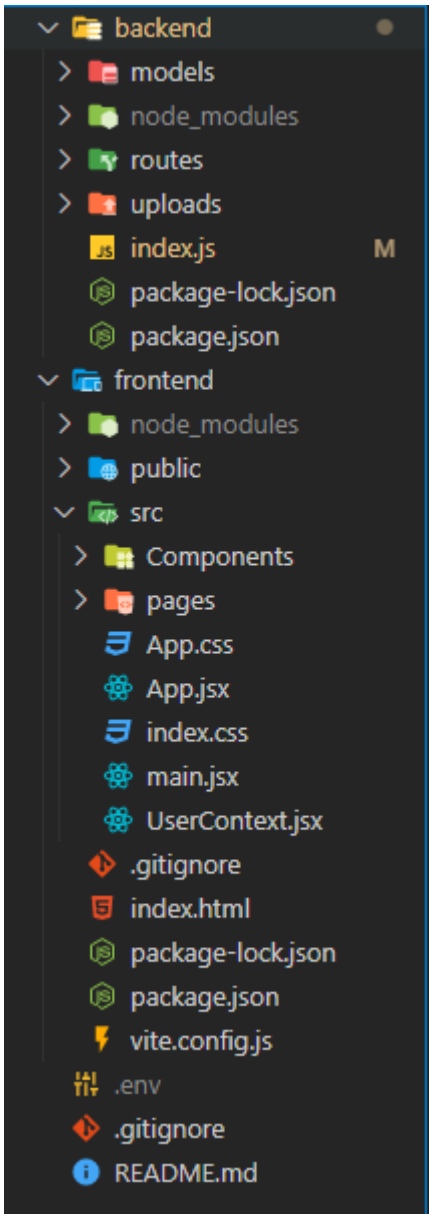


Figure 6: Folder and file structure of the application

Figure 6 above shows the screenshot of the structure of the application. There are two main folders: backend and frontend. The backend folder contains all the server-side code that are responsible for handling requests, connecting, and interacting with MongoDB and the frontend folder contains all the client-side logic responsible for rendering user interface. The models folder will be described in the 4.2.2 section, routes in 4.2.1. The uploads folder contains all the files that have been uploaded by users. The "app.jsx" file serves as the main entry point for the frontend of the application where all the components and pages are imported and setting up routing using React Router is carried out. The "UserContext.jsx" is responsible for creating and providing context to the components in the application. Then, *app.css* is a stylesheet file that contains styles of the application. And finally, "main.jsx" file initializes React application and renders "app.jsx" file.

4.2 Backend Development

The backend development is started with making a simple hello world app as in code 4 below.

```
backend > index.js > ...
1   const express = require("express");
2   const app = express();
```

Code 4: Importing express module.

First, express.js module was imported and assigned to a variable "express". Then, the other line of code creates an instance of the express application by calling express function. The app in the code normally allows to define routes and handle requests.

```
app.get("/", (req, res) => {
  res.send("hello");
});
app.listen(port, () => console.log(`server started on port ${port}`));
```

Code 5: A simple hello world example

In the above code 5, a route was defined for the root URL ("/"). When a client makes an HTTP get request, the function will be executed and the method "res.send()" sends a response back to the client as a string hello. Then, app.listen() starts the server on the specific port and console logs a message stating that the server has started on that port. After this, MongoDB is connected using Mongoose.

```
const mongoose = require("mongoose");
//connect to db
mongoose.connect(
  "mongodb+srv:/[redacted]"
);
```

Code 6: Establishing MongoDB Connection with Mongoose in Node.js

In the above code 6, mongoose module is imported that allows to work with MongoDB in Node.js application. Then, the second line of code establishes a connection to the MongoDB database using the connect() method. The argument to connect() method represents the URL of MongoDB server. The connection string has been hidden from the picture since it contains username and password of the MongoDB Atlas.

4.2.1 Routes and Route Handler

Routes and route handler work together to define the behaviour of server. Routes are specific URLs or endpoints that the users can access and interact. Hello world is an example from above figure for this where "/" is the URL or end point. Other examples are /login, /register, etc. Route handlers on the other hand are the functions that execute once a specific route is matched. Below is the screenshot of route handlers and routes that was configured for the server implementation of the application.

```
//route handlers
const loginRoute = require("./routes/login");
const registerRoute = require("./routes/Register");
const postRoute = require("./routes/post");
const profileRoute = require("./routes/profile");
const postsRoute = require("./routes/posts");
const logoutRoute = require("./routes/logout");

//configuring routes
app.use("/login", loginRoute);
app.use("/register", registerRoute);
app.use("/post", postRoute);
app.use("/profile", profileRoute);
app.use("/posts", postsRoute);
app.use("/logout", logoutRoute);
```

Code 7: Routes and Route handler

In the above code 7, the importing was carried out in the first block of code. Different route handlers were imported from routes directory. Each route handlers contains logic for handling requests for specific routes. In the second block of code, routes were configured using “app.use()” method which sets middleware for each route paths.

Below is the table that describes what each route does.

Route	Request type & Purpose
/register	POST create a user account
/login	POST user login and authentication
/post	POST/PUT create and update a post
/post/:id	GET retrieve a post by id
/posts	GET retrieve all posts
/profile	GET retrieve user profile information
/logout	POST logout user

Table1: List of routes, their purpose and HTTP request type.

Finally, all the endpoints are accessed in index.js. The main entry file for the server looks like below.

index.js

backend > index.js > ...

```
const express = require("express");
const cors = require("cors");
const app = express();
const mongoose = require("mongoose");
const port = 5000;

//route handlers
const loginRoute = require("./routes/login");
const registerRoute = require("./routes/Register");
const postRoute = require("./routes/post");
const profileRoute = require("./routes/profile");
const postsRoute = require("./routes/posts");
const logoutRoute = require("./routes/logout");

const cookieParser = require("cookie-parser");

app.use(express.json());
app.use(cookieParser());
app.use("/uploads", express.static(__dirname + "/uploads"));

//remove cors error
app.use(cors({ credentials: true, origin: "http://127.0.0.1:5173" }));

mongoose.set("strictQuery", true);
//connect to db
mongoose.connect(
  "mongodb+srv:/[redacted]
);

//configuring routes
app.use("/login", loginRoute);
app.use("/register", registerRoute);
app.use("/post", postRoute);
app.use("/profile", profileRoute);
app.use("/posts", postsRoute);
app.use("/logout", logoutRoute);

app.get("/", (req, res) => {
  res.send("hello");
});
app.listen(port, () => console.log(`server started on port ${port}`));
```

Code 8: Server Implementation

4.2.2 Models

Models were created to define the structure and relationships of the data stored in the database. For this application, there were two models created: user and post. The user model contains user data of the app while post model consists of post data. Additionally, user model handles tasks such as creating new user records, getting user information, and updating user data. Post model is responsible for creating new posts, retrieving post information, updating, and deleting it.

User.js

backend > models > User.js > UserSchema

```
1   const mongoose = require("mongoose");
2
3   const UserSchema = new mongoose.Schema({
4     username: {
5       type: String,
6       required: true,
7       unique: true,
8     },
9     password: {
10      type: String,
11      required: true,
12    },
13  });
14
15  module.exports = mongoose.model("User", UserSchema);
```

Code 9: User Model

Code 9 above shows the structure for the user data and creates a user model that interacts with user collection in MongoDB. First, mongoose module is required and then a user schema is created using the "mongoose.Schema" method. It has two fields: username and password. Finally, user model is created using "mongoose.model" method and exported. An example of user data from database once a user sign up is shown below in figure 7.

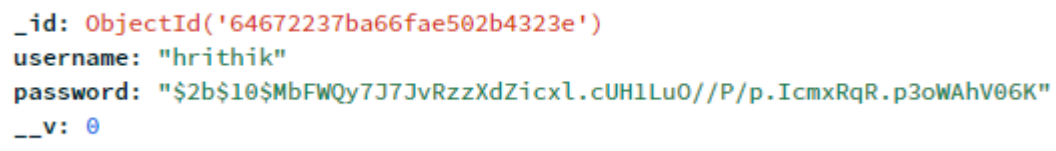


Figure 7: Screenshot of user data from database

After that comes the post model. Similarly, a post schema is created using the same method "mongoose.Schema". The schema includes fields for a post such as title, summary, photo, content, and author. Post model created for this application can be seen in code 10 below.

Post.js

backend > models > Post.js > ...

```
1   const mongoose = require("mongoose");
2   const { Schema, model } = mongoose;
3
4   const PostSchema = new mongoose.Schema(
5     {
6       title: String,
7       summary: String,
8       content: String,
9       photo: String,
10      author: { type: Schema.Types.ObjectId, ref: "User" },
11    },
12    {
13      timestamps: true,
14    }
15  );
16
17  module.exports = mongoose.model("Post", PostSchema);
```

Code 10: Post Model

Figure 8 below represents an example of a post data after a user successfully creates a post.

```
_id: ObjectId('64672390ba66fae502b43249')
title: "JavaScript Basics"
summary: "How to Work with Strings, Arrays, and Objects in JS"
content: "<p><strong>JavaScript is a popular programming language that 78% of de…"
photo: "uploads/d42f0dbaaafc24b3e38eaaf08e143455.png"
author: ObjectId('64672237ba66fae502b4323e')
createdAt: 2023-05-19T07:21:52.518+00:00
updatedAt: 2023-05-19T07:21:52.518+00:00
__v: 0
```

Figure 8: Screenshot of a post document from database

4.3 Frontend Development

In the frontend part of the thesis, user interface was created. There are components and pages such as header, posts, homepage, login page, register page, create post page, edit page, and others developed for the blog application. This section will focus on providing an overview of logic of some of the pages. Later in the result and discussion section, other components and pages will be presented and discussed.

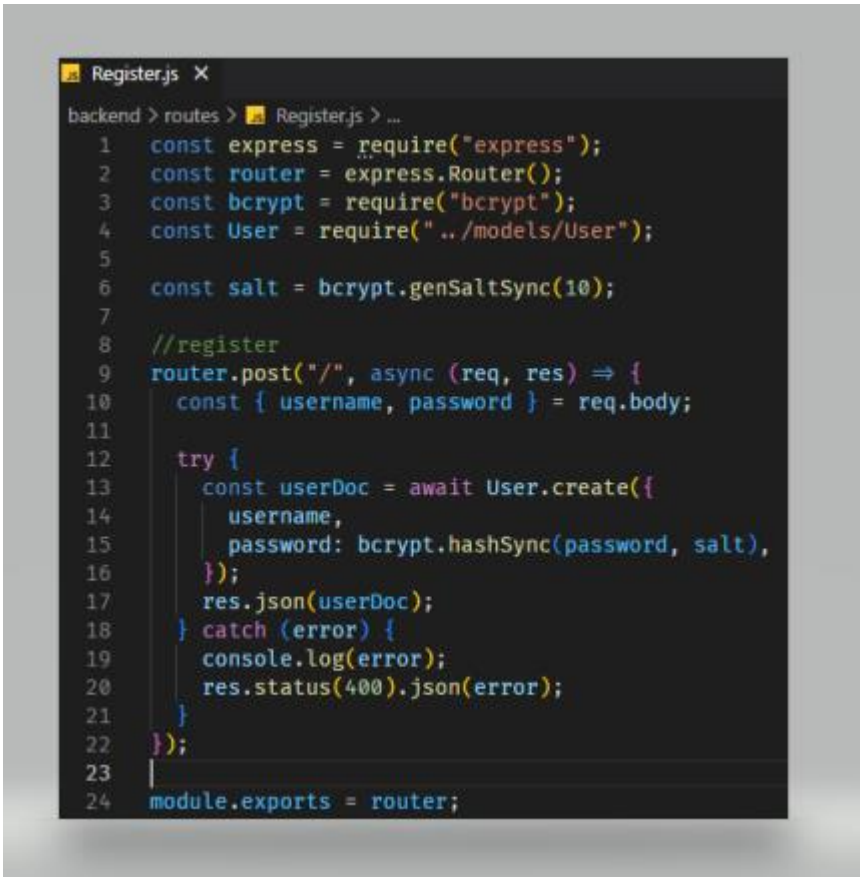
4.3.1 Register Page

```
const Register = () => {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");

  const register = async (e) => {
    e.preventDefault();

    const rsp = await fetch("http://localhost:5000/register", {
      method: "POST",
      body: JSON.stringify({ username, password }),
      headers: { "Content-Type": "application/json" },
    });

    // console.log(rsp);
    if (rsp.status !== 200) {
      alert("Registration failed");
    } else {
      alert("Registered!");
    }
  };

  return (
    <form className="register" onSubmit={register}>
      <h1>
        Register <FontAwesomeIcon icon={faUserPlus} />
      </h1>
      <input
        type="text"
        placeholder="username"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
      />
      <input
        type="password"
        placeholder="password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
      />
      <button>Register</button>
    </form>
  );
};
```

Code 11: Register Page

Code 11 above shows the register page. It renders a registration form with two fields: username and password. React hook “useState” was used to manage the state of the username and password values. Register function was added to the on submit event handler. The register function sends a POST request to the endpoint /register using fetch that includes username and password in the request body. Register /register endpoint is the same endpoint that was discussed in the route handler section. After that, an

alert message is displayed with the message “Registered!” if the response received is 200. However, an alert message “Registration failed” is received if the status code is not 200. Furthermore, the input fields were connected to the state using “value” and “onChange”. Additionally, font awesome icon “faUser-Plus” was used too. Below is the route handler for the end point mentioned above.



Code 12: Register Route Handler

Code 12 shows the route handler for register endpoint. When the form in the previous screenshot is submitted, it sends a POST request to the backend server with /register endpoint along with username and password data. From the above figure, salt is generated using “Bcrypt.genSaltSync(10)” which is used for security. Then, inside the route handler, username and password was extracted from the body which was sent after submitting the form. After that, a new user was created using “User.create()” with the submitted username and password. The “Bcrypt.hashSync()” was used to generate a hash password. Finally, the logic responded with the user document if it was successful. Otherwise, it sent a 400-status code with error message.

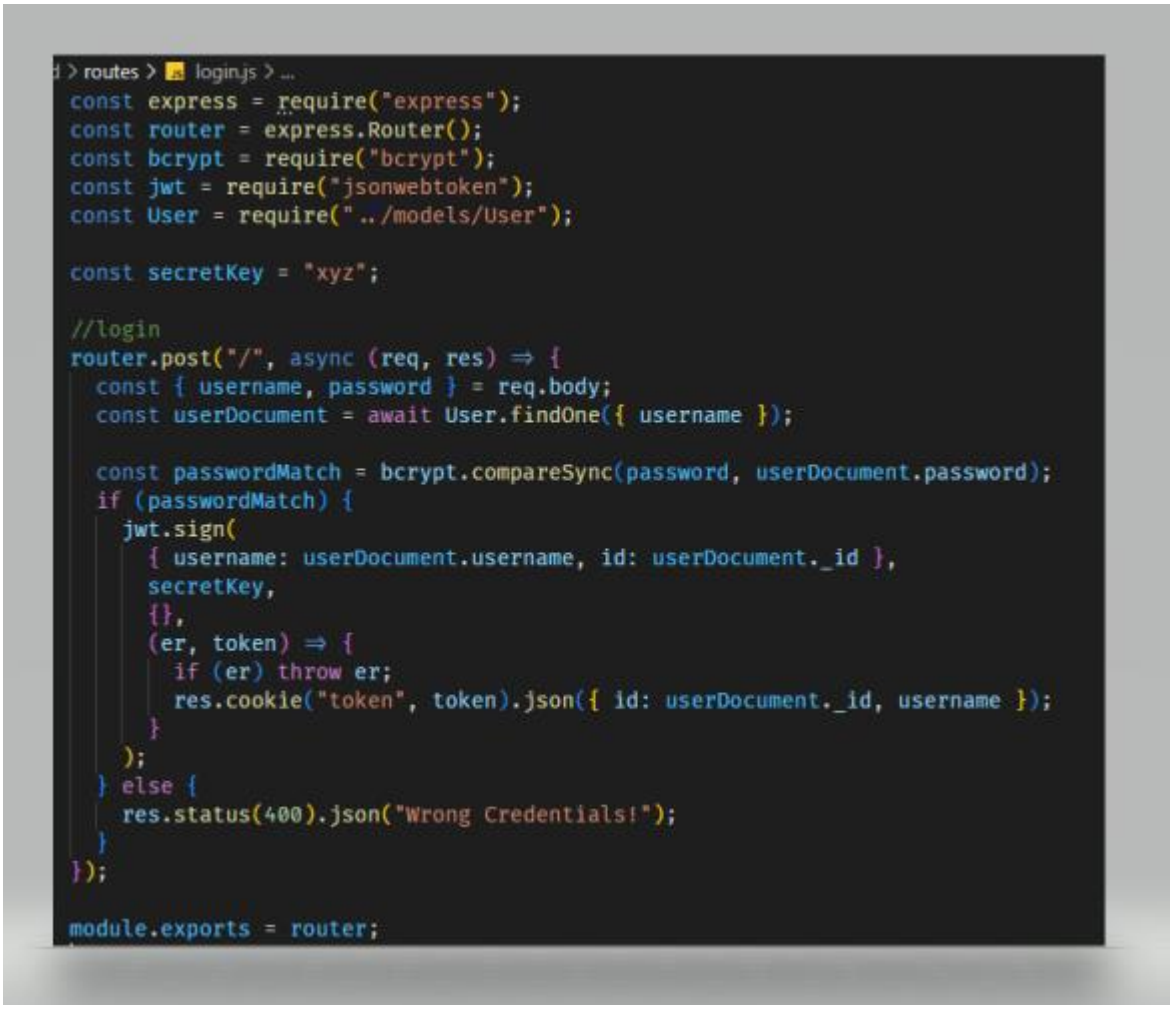
4.3.2 Login Page

```
> src > pages > LoginPage.jsx > ...
const LoginPage = () => {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [redirect, setRedirect] = useState(false);
  const { setUserInfo } = useContext(UserContext);

  // const navigate = useNavigate();
  const login = async (e) => {
    e.preventDefault();
    const response = await fetch("http://localhost:5000/login", {
      method: "POST",
      body: JSON.stringify({ username, password }),
      headers: { "Content-Type": "application/json" },
      credentials: "include",
    });
    console.log(response);
    if (response.ok) {
      response.json().then((userInfo) => {
        setUserInfo(userInfo);
        setRedirect(true);
      });
    } else {
      alert("Wrong Credentials!");
    }
  };
  if (redirect) {
    return <Navigate to={"/"} />;
  }
  return (
    <form className="login" onSubmit={login}>
      <h1>
        Login <FontAwesomeIcon icon={faArrowRightToBracket} />
      </h1>
      <input
        type="text"
        placeholder="username"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
      />
      <input
        type="password"
        placeholder="password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
      />
      <button>login</button>
    </form>
  );
};
```

Code 13: Login Page

Code 13 represents a login page for the application. The login page interacts with the server by sending a login request to the /login endpoint. The code uses React hooks such as “useState” and “useContext”. Then, a login function was defined that handles login process. The function sends a POST request to endpoint /login when the form is submitted. The request consisted of the “username” and “password” in

the request body and credentials was set to include which means that the cookies are sent along with the request. Then, if the request was successful, user information was sent to “setUserInfo”. Again, if response was ok, this triggers the redirect to run which navigates to homepage using navigate component. However, if the response was unsuccessful, an alert message is displayed with the message “wrong credentials”. The login page renders a form element which consisted of username and password as input field along with a button for submission. The “onChange” event was used for handling the input field values of “username” and “password”. Finally, login function was called once the form was submitted.



Code 14: login Route Handler

The above code 14 shows the login route handler that handles login functionality once a login request sent to the endpoint /login is received. In the figure, modules such as express, Bcrypt, JWT, and user model was imported. Then, a secret key was defined which is useful for signing JWT. After that, username and password were grabbed from request body. find() method was used in User model to find a user in the database and stored in user document variable. Moreover, received password was compared

with the hashed password that was stored in the user document with the help of “bcrypt.compareSync()”. If the password match, JWT is generated using along with payload username and id from the user document. Then, a cookie named “token” was set and JSON was sent as a response containing id and username. On the other hand, a status of 400 with a message “wrong credentials” was sent as a response if the password did not match.

5 RESULT AND DISCUSSION

The results of the MERN application developed as a part of the thesis project will be presented and discussed in this section. The application aims to provide functionality for users to register, login, create posts, and edit post. The screenshots of the final application, specifically the homepage, login page, register page, single post, mobile responsive page as well as the posts made by users will be shown and explained.

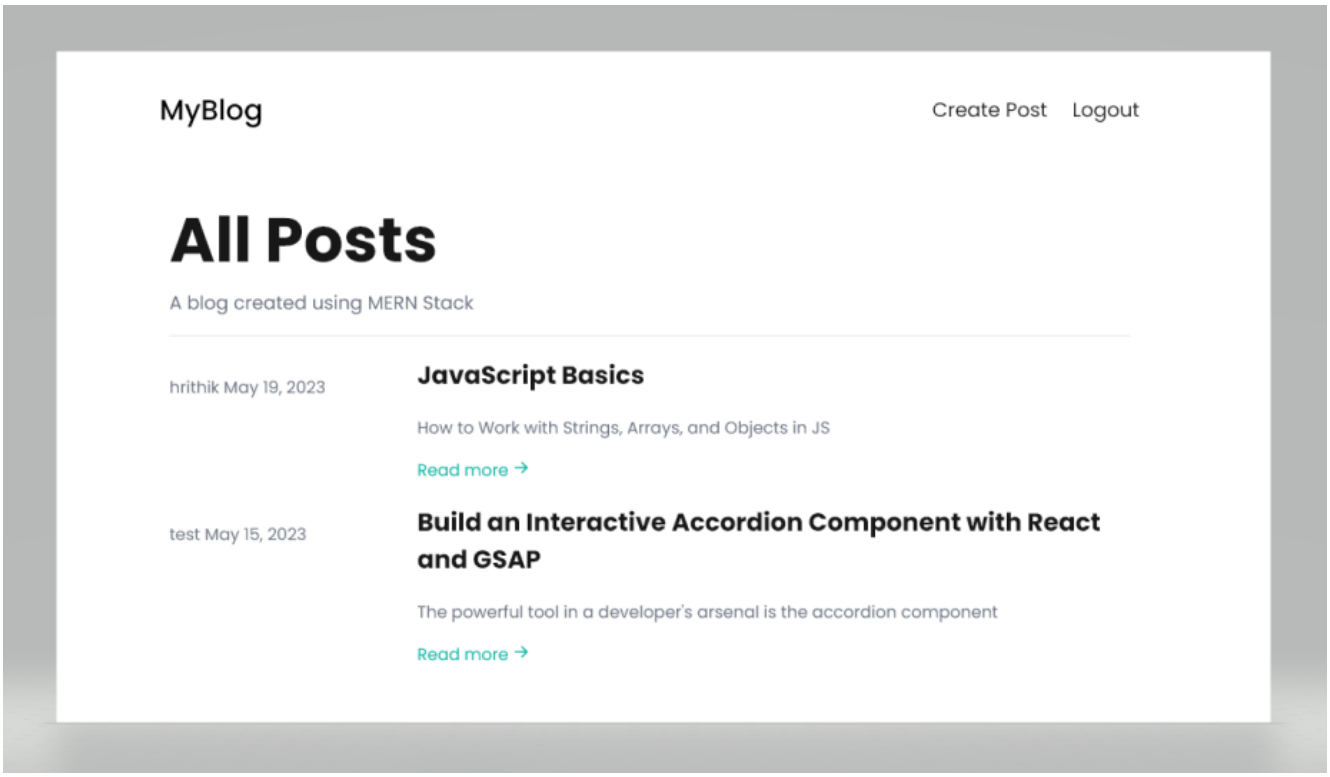


Figure 9: Home Page of Application

The above figure 9 is the homepage of the application. The header section provides user with access to create a post and logout functionalities. Below the header section, there are all the posts made by all the users. There are two users with one post each in the application. Additionally, the homepage includes name of the author and timestamp feature to display the dates when each post was made.

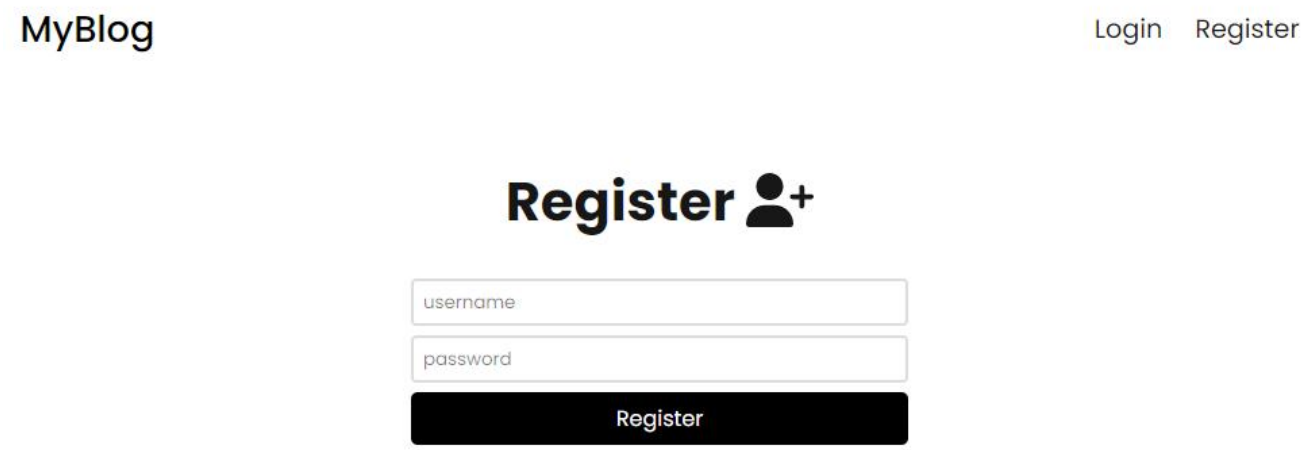


Figure 10: Register Page

Figure 10 shows the register page. Users can enter username of their choose and add password and then click on register button. After that, an alert message will be displayed with the message registered after successful registration and then the user can login with the selected username and password.

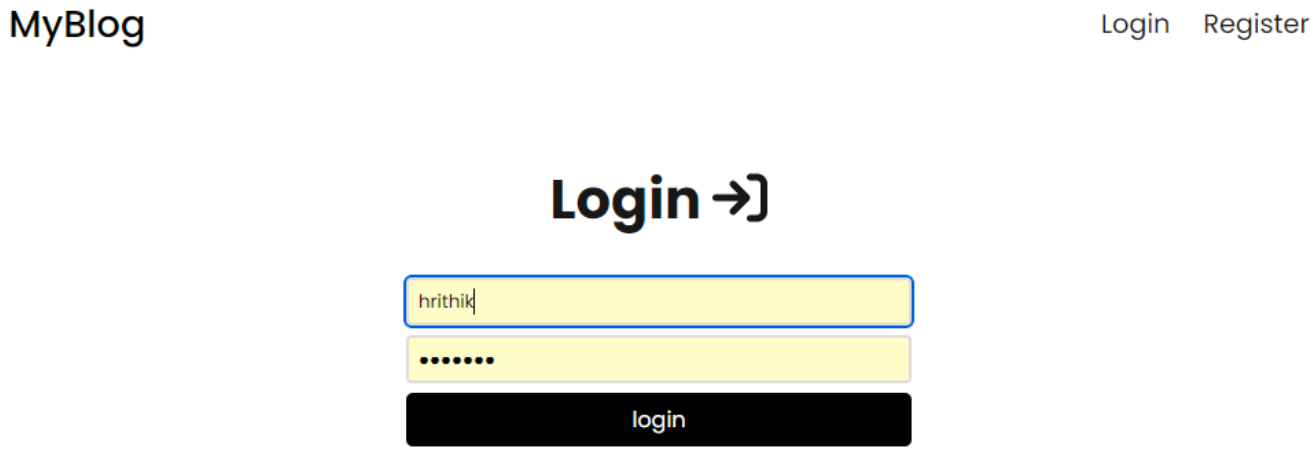


Figure 11: Login Page

Figure 11 represents the login page of the application. Users can entry their username and password in this page. After successfully entering the details and clicking login button, users will be directed to the homepage.

Similarly, if we click on one of the posts, for example, the second post from figure 21 about building an interactive accordion component with React and GSAP. We will be directed to a single post. Below is the screenshot of a single post.

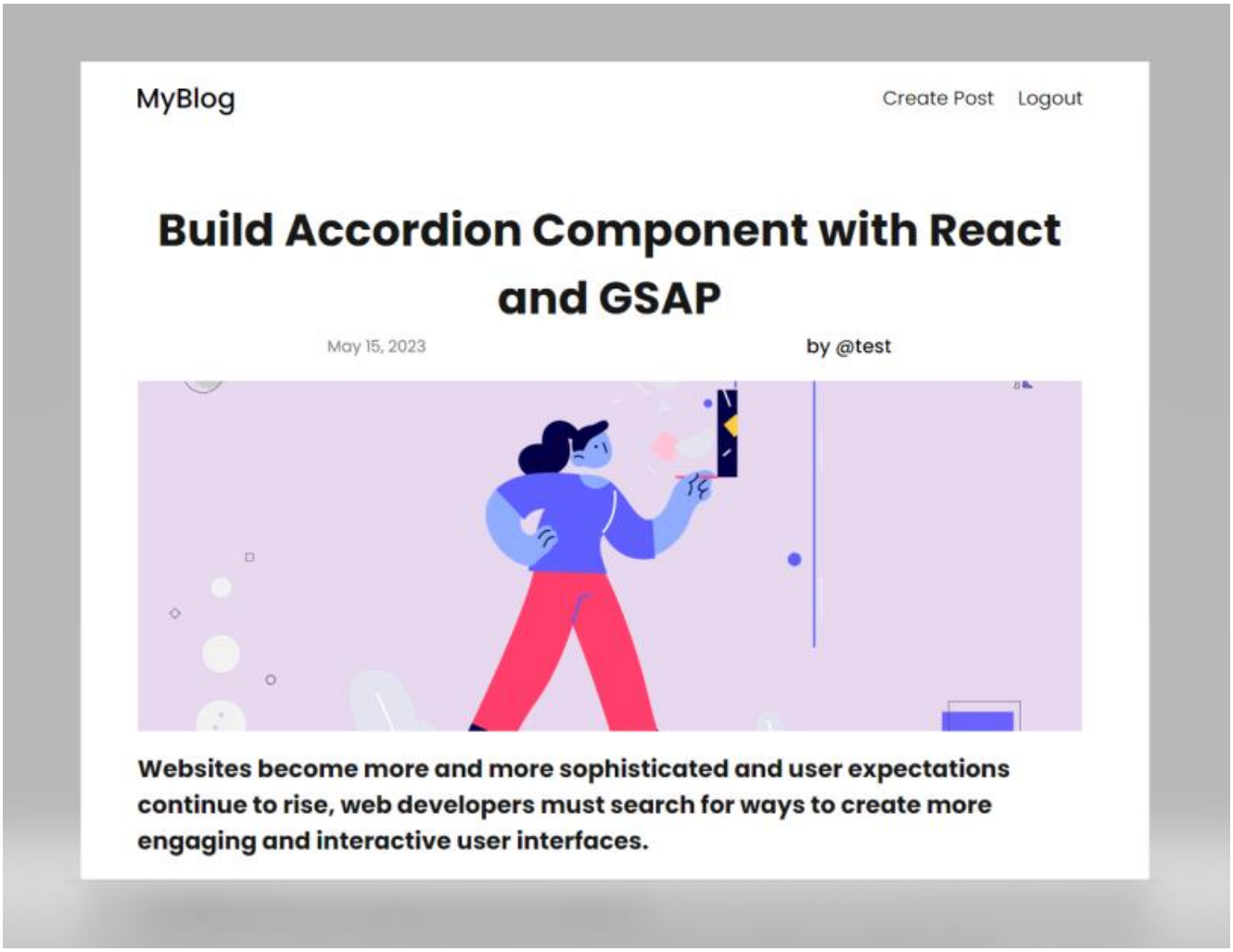


Figure 12: Single Post

Figure 12 shows a single post by user “test” posted in a specific date. Currently, the user is “Hrithik”, so this user does not have the functionality of editing the post. Only the author of the post can edit the post. Therefore, this post can only be edited by the user “test”.

The blog application is fully responsive across different screen sizes ensuring optimal user experience across various screen sizes. As an example, below is the screenshot of the homepage and a single post of the project in mobile.

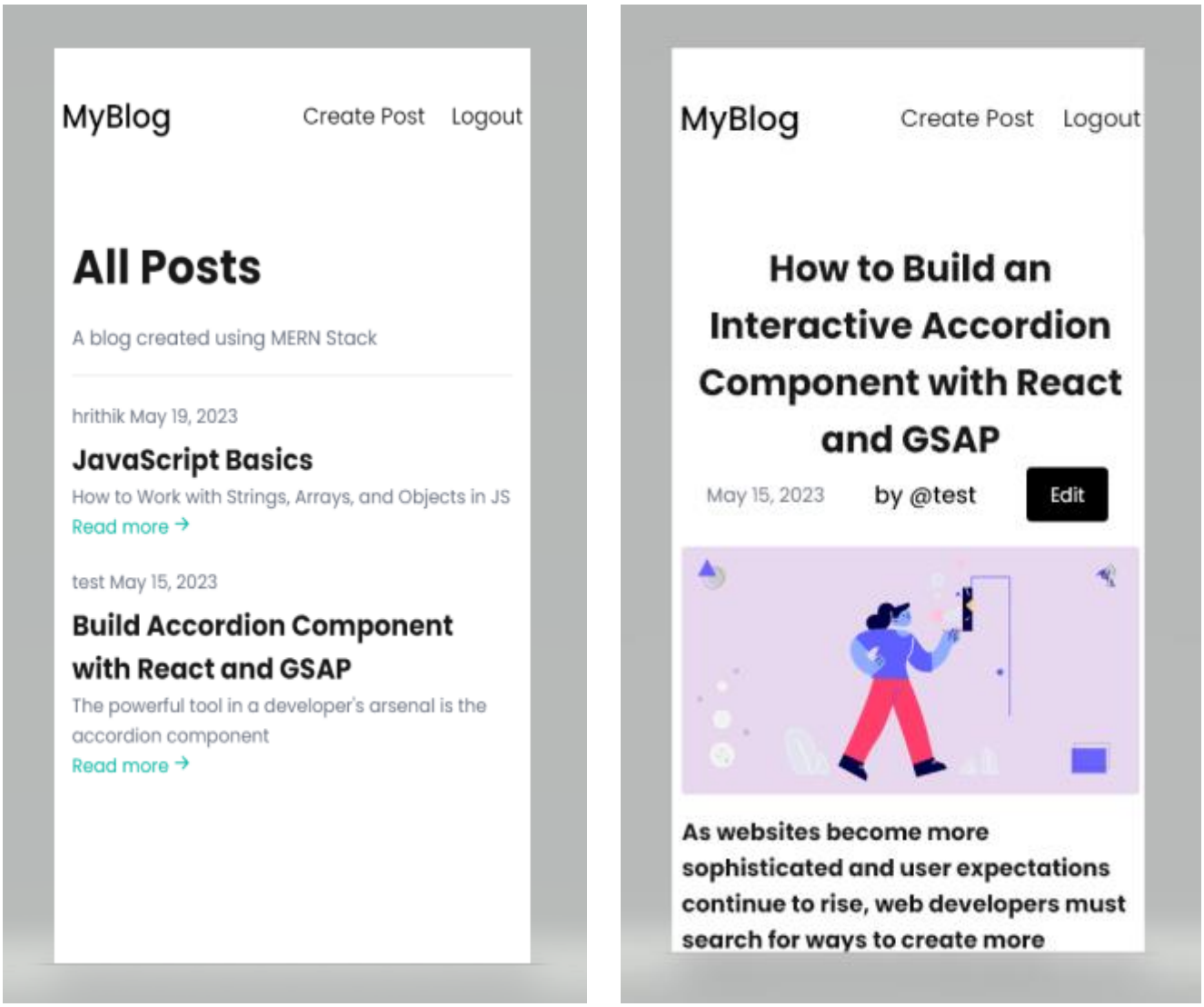


Figure 13: Mobile Responsive

Figure 13 shows the mobile responsive version of the application.

Overall, the application developed for this thesis project comprises of several essential functionalities. While these features provide a solid foundation for a blog application, there are several potential improvements that could enhance its functionality and user experience. The application does not have a delete functionality. So, adding a delete post feature would allow users to remove their posts or contents that they no longer wish to display. This feature would provide users with control over their posts. Similarly, having a search functionality would be a great addition in this project. Including an input field to search for posts would enable users to quicky locate specific content. As the number of users and posts grows, this functionality would improve the usability of the blog application.

Adding tags to posts would be a good feature to implement. This would allow users to categorize their posts based on the content. For example, if a post is related to JavaScript, there would be a tag named JavaScript which would then be used to filter and explore topics related to it. Then, a page to manage user profile would allow users to change their details such as profile picture, bio, and any other useful information they may want to add. In the application, there is no feature to view all the post posted by a user in a separate page. The user is only able to view all the posts in the homepage along with posts from different users. When the volume of the posts rises, there might be confusion and difficulty in finding own posts. Thus, having a separate page for all the posts published could also be a feature to implement.

The discussed ideas for improving the blog application are significant that can be considered for the future implementation. Once the new features and improvements have been added and tested, the application could be deployed in the production mode. Deployment would enable users to access this application and make use of it.

6 CONCLUSION

The thesis aimed to explore the theoretical foundations of MERN and build a blog application from scratch using MERN stack. By acquiring a deep understanding of each of the components of the MERN stack, a functional user-friendly blog application was successfully developed that enables users to create and edit posts.

Throughout the development of the application, several functionalities were implemented. Users can register, login securely, browser existing posts, create new posts, make edits, and logout when desired. Moreover, the blog application was designed to be responsive across various screen sizes and devices to ensure consistent and accessible experience for users.

In addition to the achievements of developing the blog application, the thesis also emphasized the need for improvements. The discussion section of thesis highlighted various areas in the application for further improvements which include delete post functionality, search field, adding user profile and tags, and separate page for viewing own posts. These ideas could be used for expanding the functionalities of this application which could promote user engagement.

REFERENCES

Coremans, C. 2015. *HTML: a beginner's tutorial*. Brainy Software, Inc.

Dean, J. 2019. *Web programming with HTML5, CSS, and JavaScript*. Burlington, Massachusetts: Jones & Barlett Learning.

Haviv, A.Q. 2016. MEAN Web Development. Packt Publishing.

Hoque, S. 2020. *Full stack React projects*: *Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node.js.* Birmingham: Packt Publishing.

Java T Point. 2011-2021. *MERN Stack*. Available: https://www.javatpoint.com/mern-stack. Accessed on January 28,2023.

MongoDB. 2023. *How to use MERN stack: A complete guide*. Available: https://www.mongodb.com/languages/mern-stack-tutorial . Accessed on February 20, 2023.

MongoDB. 2023. *Documentation – Full Stack Development Explained*. Available at: https://www.mongodb.com/languages/full-stack-development. Accessed 10 January 2023.

MongoDB.2023. *Documentation – What is MongoDB.* Available at: https://www.mongodb.com/what-is-mongodb . Accessed 10 January 2023.

Microsoft. 2023. *Documentation – Get Started with Visual Studio Code*. Available at: https://code.visualstudio.com/learn . Accessed 16 May 2023.

Meta Open Source. 2023. *Documentation - React*. Available at: https://react.dev/ . Accessed 22 May2023.

Meta Open Source. 2023. *Documentation - Learn your first component*. Available at: https://react.dev/learn/your-first-component . Accessed 15 May 2023.

Meta Open Source. 2023. *Documentation - Learn writing markup with JSX*. Available at: https://react.dev/learn/writing-markup-with-jsx . Accessed 18 May 2023.

Meta Open Source. 2023. *Documentation - Learn using hooks*. Available at: https://react.dev/learn#using-hooks . Accessed 22 May 2023.

Meta Open Source. *2023. Documentation - Learn passing props to a component*. Available at: https://react.dev/learn/passing-props-to-a-component . Accessed 20 May 2023.

MDN Web Docs. 2019. JavaScript basics. Available at: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics#what_is_javascript . Accessed 12 May 2023.

Matti, L.2023. *Fullstack-part3 | Saving data to MongoDB* . Available at: https://fullstackopen.com/en/part3. Accessed 20 May 2023.

Matti, L.2023. *Fullstack-part3 | Node.js and Expres* . Available at: https://fullstackopen.com/en/part3. Accessed 20 May 2023.

Matti, L.2023. *Fullstack-part7 | React Router* . Available at: https://fullstackopen.com/en/part7 . Accessed 20 May 2023.

Matti, L.2023. *Fullstack-part1 | Introduction to React* . Available at: https://fullstackopen.com/en/part1/introduction_to_react . Accessed 20 May 2023.

Matti, L.2023. *Fullstack-part1 | JavaScript* . Available at: https://fullstackopen.com/en/part1/introduction_to_react . Accessed 19 May 2023.

Northwood, C. 2018. The Full Stack Developer: Your essential guide to the everyday skills expected of a modern full stack web developer. Apress.

Nirgudkar, N. & Singh, P .2017. The MEAN Stack. Available at: https://www.irjet.net/archives/V4/i5/IRJET-V4I5795.pdf . Accessed 10 May 2023.

NPM. 2023. *Documentation - About NPM*. Available at: https://docs.npmjs.com/about-npm . Accessed 20 May 2023.

NPM. 2023. *Documentation - Nodemon*. Available at: https://www.npmjs.com/package/nodemon . Accessed 25 May 2023.

NPM. 2017. *Documentation - Bcrypt JS*. Available at: https://www.npmjs.com/package/bcryptjs . Accessed 25 May 2023.

NPM. 2022. *Documentation - Multer*. Available at: https://www.npmjs.com/package/multer . Accessed 25 May 2023.

Subramanian, V. 2019. Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node. Berkeley, Ca: Apress.

Stack Overflow. 2022. *Stack Overflow Developer Survey 2022*. Available at: https://survey.stackoverflow.co/2022/#most-popular-technologies-language . Accessed 10 April 2023.

Serdar, Y. 2018. *What is the MEAN stack? JavaScript web applications*. Available at: https://www.proquest.com/docview/2133657133?accountid=10007 . Accessed 5 May 2023.

Sharma, M. 2022. *Full Stack Development with MongoDB*. BPB Publications.

Vassallo, K. & Garg, L. 2016. Cross-platform development frameworks: overview of contemporary technologies and methods for cross-platform application development. Available at: https://www.um.edu.mt/library/oar/handle/123456789/25282 . Accessed 1 May 2023.

Vite. (n.d). *Getting Started | Vite*. Available at: https://vitejs.dev/guide/ . Accessed 10 May 2023.