VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Tran Anh

Building a Prototype Web Application for Travelers to Share Camping Sites

Technology and Communication

2023

ABSTRACT

Author	Tran Anh
Title	Building a Prototype Web Application for Travelers to Share Camping Sites
Year	2023
Language	English
Pages	51
Name of Supervisor	Kenneth Norrgård

The motivation to build this project was to establish a social platform that connects individuals who share a passion for camping. This is a good place for campers to reference and compare different camping sites. In this project, several technologies like ReactJS and TypeScript are used to build with Bootstrap for a professional outlook and Nodejs, Express with Mongoose for the backend and database.

To design and develop a responsive website for users to share and post their experiences about campgrounds. To implement features such as user account creation, CRUD functionality for campground posts, authentication for post updates/deletions, and the ability for users to leave comments. To evaluate the performance and usability of the website and make recommendations for future improvements.

Keywords JavaScript, React, NodeJS, MongoDB, Bootstrap

ACKNOWLEDGEMENT

I express my sincere gratitude to my supervisor, Mr. Kenneth Norrgård, for his invaluable support and guidance during my thesis project and thesis writing. His expertise and mentorship have been instrumental in my academic journey.

I would also like to extend my appreciation to VAMK, University of Applied Sciences, where I have had the privilege of studying for four years. The knowledge and skills I have gained from my time at VAMK are truly valuable and will continue to benefit me in my future career.

Last but not least, I want to express my heartfelt appreciation to my family and friends for their support throughout my university studies. Their love and encouragement have been a constant source of motivation for me, and I am grateful for their presence in my life.

Tran Anh,

Vaasa,Finland,2023

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

LIST OF ABBREVIATIONS

API	Application program interface
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	Hypertext Transfer Protocol
MERN	MongoDB, ExpressJS, ReactJS, NodeJS
VDOM	Virtual DOM
URL	Uniform Resource Locator
SQL	Structured Query Language
IDE	Integrated development environment
npm	node-package-manage.
UI	User Interface
JS	JavaScript
I/O	Input/Output

1 INTRODUCTION

1.1 Project Overview

Finding a new place for camping can be a challenge for campers and outdoor enthusiasts. To simplify this process, a web application can be developed to assist users in finding and selecting the best camping locations. The website will provide a platform for campers to share their experiences and knowledge about different campgrounds, making it easier for other campers to plan their trips and find the best campground for their needs.

This report will outline the process of creating a modern, user-friendly web application for campers and outdoor enthusiasts. To achieve this goal, the Bootstrap library of UI components will be utilized to build a responsive and user-friendly web application.

The design of the website will be based on user needs and expectations to ensure that the website is user-friendly the website is used popular programming languages lead to easy to scale up the website will be deployed for a user to access and give feedback.

1.2 Objective

In this application, users will have the ability to share and post their favourite camping sites, providing other campers with the opportunity to experience these locations first-hand. This feature will enable users to contribute to the community and help others discover new and exciting camping destinations. To make the posting process as user-friendly as possible, the application will provide a simple and intuitive interface for users to upload photos, write descriptions, and provide details about the campground.

2 BACKGROUND AND PURPOSE OF THE PROJECT

The background and purpose of the camping site prototype project are centered around the need for a platform that allows individuals to easily find and share information about suitable camping locations. The main goal of the project is to create a prototype application that provides a user-friendly website where camping enthusiasts can register as members and contribute by sharing their experiences about campgrounds they have personally visited.

The project provides a comprehensive database of camping locations, complete with detailed information about price, location and user reviews. This information will assist campers in making informed decisions when selecting a campground for their next camping trip. By facilitating the exchange of first-hand experiences and insights, the website creates a supportive community of campers who can share their knowledge and expertise with fellow outdoor enthusiasts.

There might be a different viewpoint and method to build this application, but which will not be covered in this thesis. Cybersecurity and search optimization, for example, are crucial aspects that need to be considered when developing any software system. Although these aspects are important, with the limit time and in the scope of the thesis we are not able to handle all of these aspects.

3 THEORETICAL BACKGROUND

In the field of web development, a 'stack' refers to a combination of different technologies used to build a web application. This Bachelor thesis, I focuses on building a social platform that connects individuals who share a passion for camping. To achieve this goal, I chose to use the MERN stack a popular and powerful web stack that includes MongoDB, Express, React, and Node.js.

In this section of my thesis, I will provide a detailed overview of the MERN stack and explain why it was the ideal choice for my camping platform. Additionally, I will discuss other technologies that were used to manage important aspects of the application, such as state management and authentication. By the end of this chapter, readers will have a clear understanding of the technologies used in my project and the reasons behind their selection.

3.1 Front-end

The development of the camping site application involved a significant amount of time and effort in coding. The front-end development took approximately 5 weeks, during which I worked on designing and implementing the user interface and user experience of the website.

3.1.1 ReactJS

ReactJS is a popular JavaScript library for building user interfaces. It was developed by Facebook and released in 2013. ReactJS provides a declarative programming model that simplifies the process of building complex user interfaces by breaking them down into smaller, reusable components (Kumar Ray,2023).

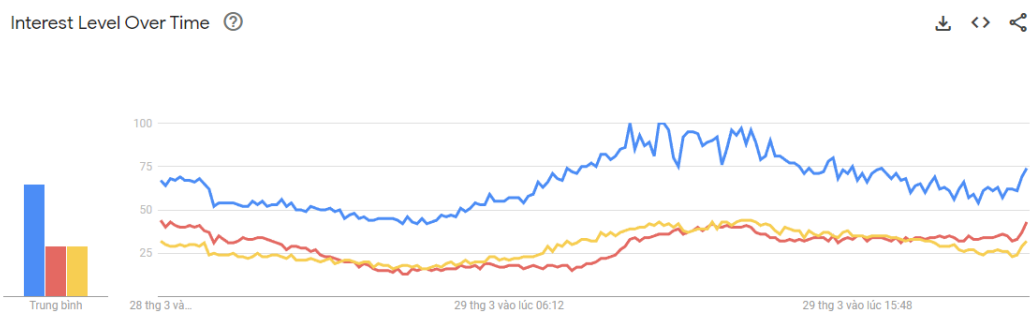


Figure 1 React vs Vue vs Angular search trend 2023 (Google Trends,2023).

One of the key benefits of ReactJS is its use of a virtual DOM, which allows for efficient updates to the user interface in response to changes in data. Rather than updating the entire DOM tree whenever data changes, ReactJS only updates the parts of the tree that have changed, making it fast and efficient (Kumar Ray,2023).

ReactJS also has a large and active community, with many resources available for learning and troubleshooting. There are numerous online tutorials, documentation, and forums available for developers to learn from and share their knowledge.

For my camping platform project, ReactJS was chosen because of its ease of use, efficiency, and popularity. By using ReactJS, it was possible to create reusable UI components that could be used throughout the application, saving time and effort in the development process.

```jsx
const Footer = () => {
  return (
    <footer className="footer py-3 mt-auto">
      <div className="container">
        <span className="text-light">&copy; TranAnh 2023</span>
      </div>
    </footer>
  );
};

function App() {
  return (
    <div className="App">
      <Footer />
    </div>
  );
}
```

Figure 2 Example of reusable UI component

3.1.2 HTML & CSS

HTML (Hypertext Markup Language) is a markup language used to create and structure content for the web. It defines the elements and attributes used to describe the content, including text, images, and links, and provides a standardized way for web browsers to interpret and display this content (Astari ,2023).

CSS (Cascading Style Sheets) is a style sheet language used to control the presentation and layout of HTML content. It defines styles for fonts, colors, margins, and other visual elements, allowing developers to create visually appealing and consistent web pages. Together, HTML and CSS form the backbone of modern web development, providing the tools and standards necessary to create dynamic and engaging websites (Cascading Style).

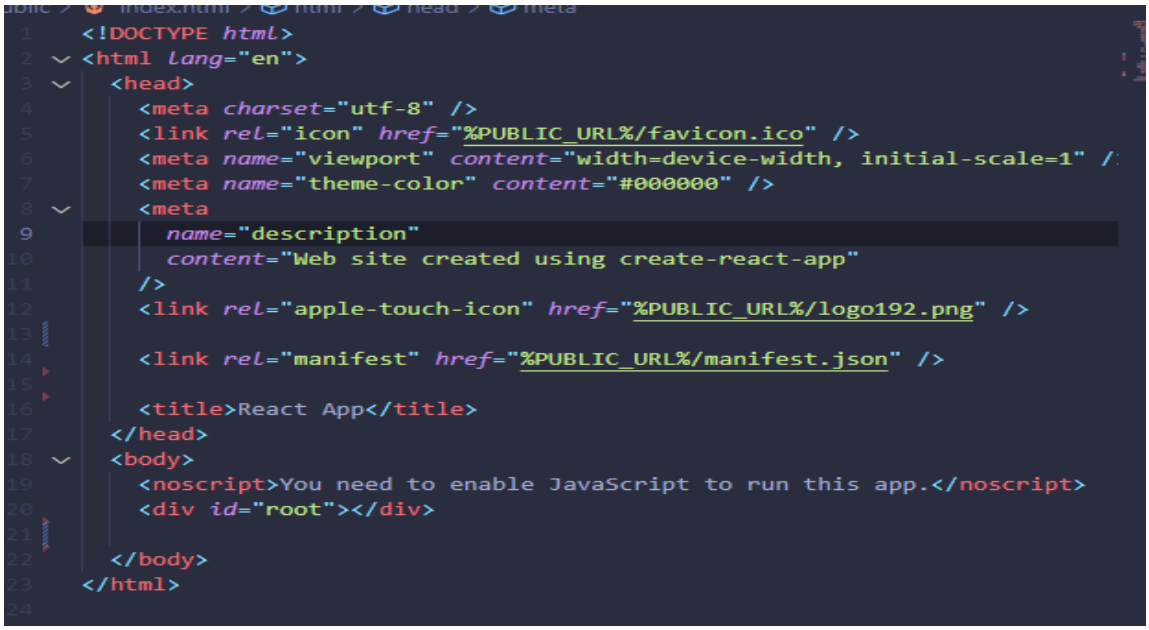


Figure 3 Example of HTML page

3.1.3 JavaScript

JavaScript is a popular, high-level programming language used to create dynamic and interactive web content. It is widely used to add functionality to websites and create interactive features such as drop-down menus, pop-up boxes, and user input forms. With its versatility and wide adoption, JavaScript has become an essential tool for front-end web developers and is often used in conjunction with other web technologies such as HTML and CSS (Williams,2022). Its popularity has also led to the development of numerous frameworks and libraries that extend their capabilities and simplify the development process for web applications.

3.1.4 Typescript

TypeScript is a superset of JavaScript that adds optional static typing and other advanced features to the language. It was developed by Microsoft and is open source, with a large and growing community of users and contributors. TypeScript is designed to improve the scalability and maintainability of large-scale JavaScript

applications, by enabling developers to catch errors and bugs at compile-time rather than runtime (Gomes,2023). Additionally, TypeScript provides features such as interfaces, classes, and modules, which can help developers organize and structure their code in a more efficient and reusable way. Overall, TypeScript is a powerful tool for building robust and reliable JavaScript applications.

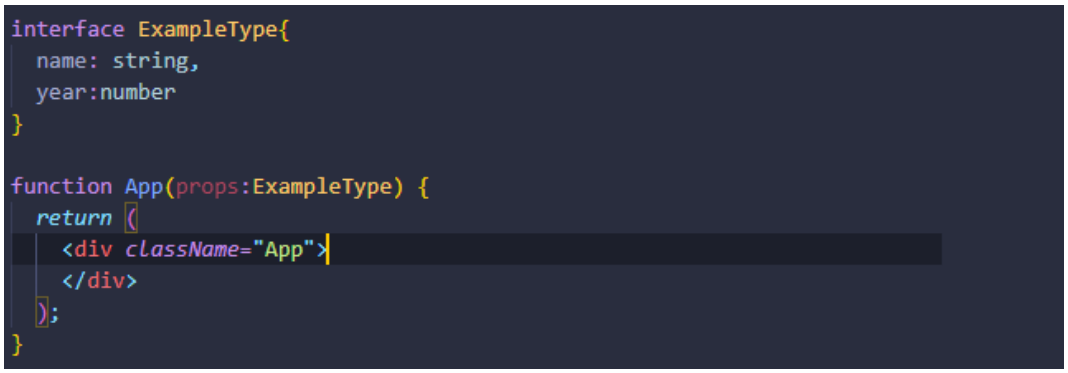


Figure 4 Example of using interface in TypeScript.

3.1.5 Redux

Redux is an open-source JavaScript library that provides a predictable state container for managing the state of a web application. It enables developers to write applications that behave consistently across different environments and scales easily as the application grows (Redux). At its core, Redux works by storing the state of the application in a single store, which can only be modified through dispatched actions. These actions describe the state changes that should occur, and the store then updates the state accordingly. By enforcing this strict separation between state and logic, Redux enables developers to build more modular and maintainable applications.

3.1.6 Redux Toolkit

Redux Toolkit is a library designed to simplify and streamline the process of building Redux-based applications. It includes a set of preconfigured tools and utilities that can help developers write code more efficiently and reduce boilerplate. Some of these tools include the createSlice function for defining Redux reducers, a standardized way of defining Redux actions, and built-in middleware for handling

common use cases like asynchronous requests. By using Redux Toolkit, developers can reduce the amount of code they need to write and maintain, while also improving the readability and maintainability of their Redux code (Redux).

3.1.7 Bootstrap

Bootstrap is a popular open-source framework for building responsive and mobile-first websites. It provides a collection of pre-built HTML, CSS, and JavaScript components that can be easily integrated into any web project, saving developers time and effort. The framework was developed by Twitter and is now maintained by a community of developers (Zola ,n.d).

Bootstrap is designed to be flexible and customizable, allowing developers to easily modify the look and feel of their website to fit their needs. It also includes a responsive grid system that makes it easy to create layouts that adapt to different screen sizes, ensuring that the website looks great on desktops, tablets, and mobile devices(Davidson,2023).

One of the main benefits of using Bootstrap is its large and active community. This community provides a wealth of resources, including documentation, examples, and support, making it easy for developers to get up and running with the framework. Additionally, because Bootstrap is widely used, it is easy to find third-party plugins and extensions that can extend its functionality.

```
<div className="toast-container position-fixed bottom-0 end-0 p-3">
  <Toast className="toast fade" autohide delay={3000} show={showAlert}>
    <div className="toast-header ">
      {status === "danger" ? (
        <AiFillWarning className="me-2" />
      ) : (
        <CiMountain1 className="me-2" />
      )}

      <strong className="me-auto">Camping Site</strong>
      <button
        type="button"
        className="btn-close"
        onClick={() => dispatch(setShow({ value: false }))}
      ></button>
    </div>
    <div className={`toast-body text-bg-${status}`}>{errorMess}</div>
  </Toast>
</div>
```

Figure 5 Example using Bootstrap.

3.2 Backend

The back-end development took approximately 3 weeks, during which the server-side of the application using Node.js and Express.js was built. This included creating and integrating different APIs, handling user authentication and authorization, and connecting to the database.

3.2.1 NodeJS

Node.js is a popular open-source JavaScript runtime environment that allows developers to execute code outside of a web browser. It is built on top of the V8 JavaScript engine from Google and uses an event-driven, non-blocking I/O model that makes it highly efficient and scalable. Node.js is commonly used for building server-side applications, such as web servers, APIs, and real-time chat applications. It provides a vast array of built-in modules and third-party packages, which allow developers to easily add features and functionality to their applications. One of the key benefits of Node.js is its ability to handle large volumes of data and requests with minimal overhead, making it a popular choice for high-performance applications (Semah,2022) .

3.2.2 ExpressJS

Express.js is a powerful web application framework for Node.js that provides developers with a robust set of tools for building scalable and efficient web applications. With its minimalist design and modular architecture, Express.js allows developers to create custom middleware and easily integrate third-party libraries, making it a popular choice for building server-side applications (Building Web Application ,n.d).

Express.js is known for its ability to handle HTTP requests and responses, routing, and middleware in a clean and organized way. Developers can use its built-in features for handling HTTP requests and responses, including URL parsing, cookie management, and error handling. Additionally, Express.js enables developers to create custom middleware to handle tasks such as authentication and logging, as well as to integrate third-party middleware for additional functionality.

Express.js is also highly extensible and has a large and active community of developers who contribute to its ecosystem of plugins and libraries. This allows developers to quickly and easily add functionality to their applications without having to write everything from scratch.

Figure 6 Example of ExpressJS

3.2.3 MongoDB

MongoDB is a popular NoSQL database that is widely used in web development. Unlike traditional SQL databases, which use tables with rows and columns to store

data, MongoDB uses a document-based model that allows for more flexible and scalable data storage (MongoDB, n.d).

In this project, MongoDB was used as a database for storing data related to camping sites, user information, and more. By using MongoDB, we were able to take advantage of features such as:

- Flexible schema design: MongoDB allows for dynamic and flexible schema design, which means that we can easily add or modify fields in our data without having to modify the entire database structure.
- Scalability: MongoDB is designed to scale horizontally, which means that we can add more servers to handle increased traffic and data volumes.
- Querying and indexing: MongoDB provides powerful querying and indexing capabilities that allow us to search and retrieve data quickly and efficiently.

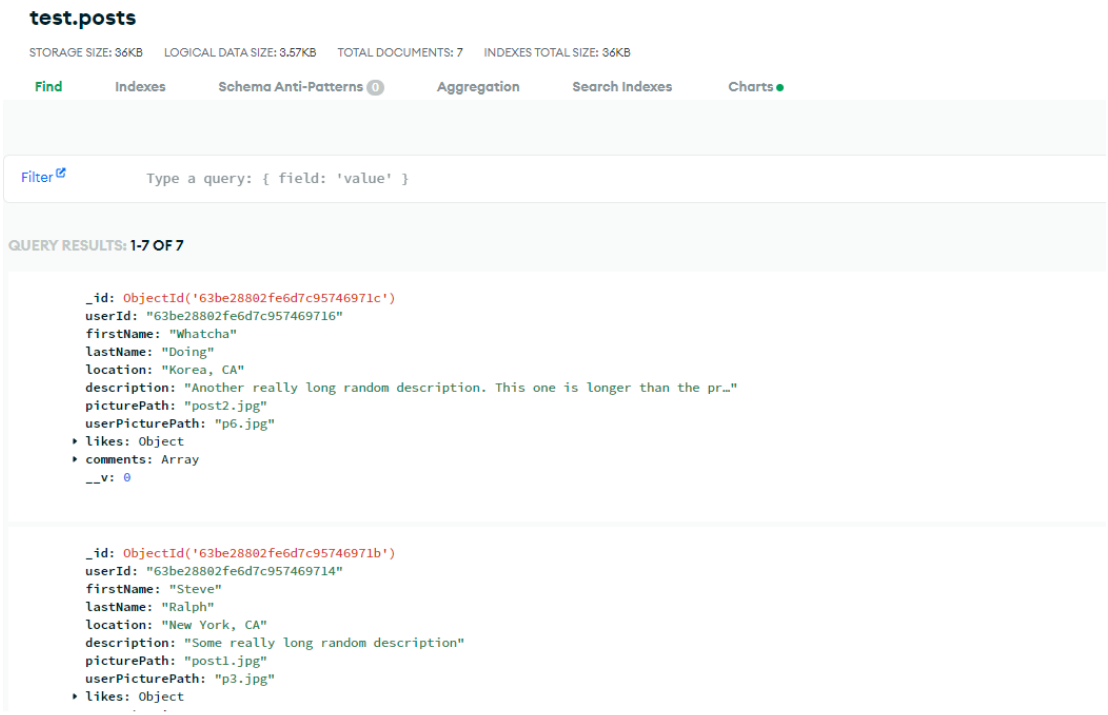


Figure 7. Example of MongoDB

4 APPROACH AND IMPLEMENTATION

In order to provide a clear and understandable overview of the project, this chapter will begin by discussing the user architecture of the camping platform. This includes a detailed explanation of the various components and features that make up the user interface, as well as the underlying technologies used to develop and implement these features.

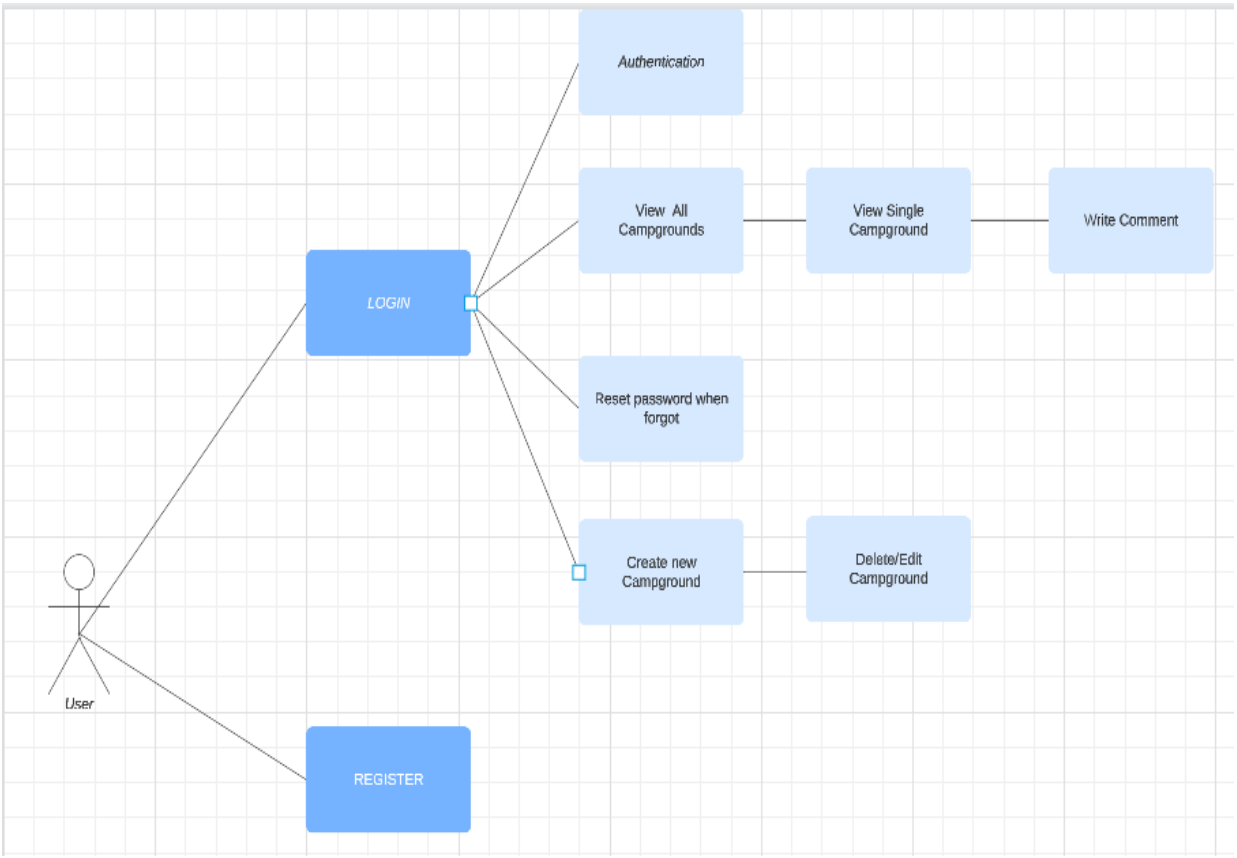


Figure 8. Use case flowchart of the project.

The development of any software project requires careful planning and organization in order to ensure that the final product meets the needs and expectations of its users. One key aspect of this planning process is the establishment of a comprehensive set of requirements that define the features and functionality of the

software. These requirements serve as a roadmap for the development process, guiding the design and implementation of the software and helping to ensure that it meets the needs of its intended users.

We present a table of requirements and priorities for our camping platform. This table serves as a detailed and comprehensive list of the features and functionality that we aim to include in the platform, as well as a clear indication of the relative importance of each requirement. By establishing these requirements and priorities at the outset of the development process, we can ensure that our platform meets the needs of our target audience and delivers a high-quality user experience.

Table 1. Table of requirements

REQUIREMENTS	PRIORITY
User registration and login functionality	1
Ability for users to create and edit camping site posts	1
Ability for users to view camping site posts	1
Ability for users to comment on camping site posts	2
Responsive design for mobile and desktop devices	2
Recover password when forgot	2
Showing the camping site on map	2

Now that we have established the requirements and priorities for our camping platform, the next step is to begin developing the software. In order to do this, we must first set up our coding environment with the necessary tools and technologies. In this section, we will outline the steps required to configure our development environment, including the installation of the required software packages and the configuration of our development environment. By following these steps,

we will be able to begin coding our platform and implementing the requirements that we have established in the previous section.

4.1 Integrated Development Environment (IDE) or Code Editor

For this project, Visual Studio Code for chosen as a code editor. The main reason for choosing VS Code is its deep built-in support for NodeJS and the broad range of programming languages it supports, such as JavaScript, TypeScript, and JSON. VS Code is known for its powerful code refactoring and debugging capabilities, along with its extensive library of extensions, plugins, and themes. These features provide developers with valuable support throughout the programming process, from code creation to deployment. Furthermore, the widespread use of VS Code in the software development community ensures that it is a well-documented and continuously updated tool. VS Code access to a robust and user-friendly platform that will enable completion of this project efficiently and effectively.

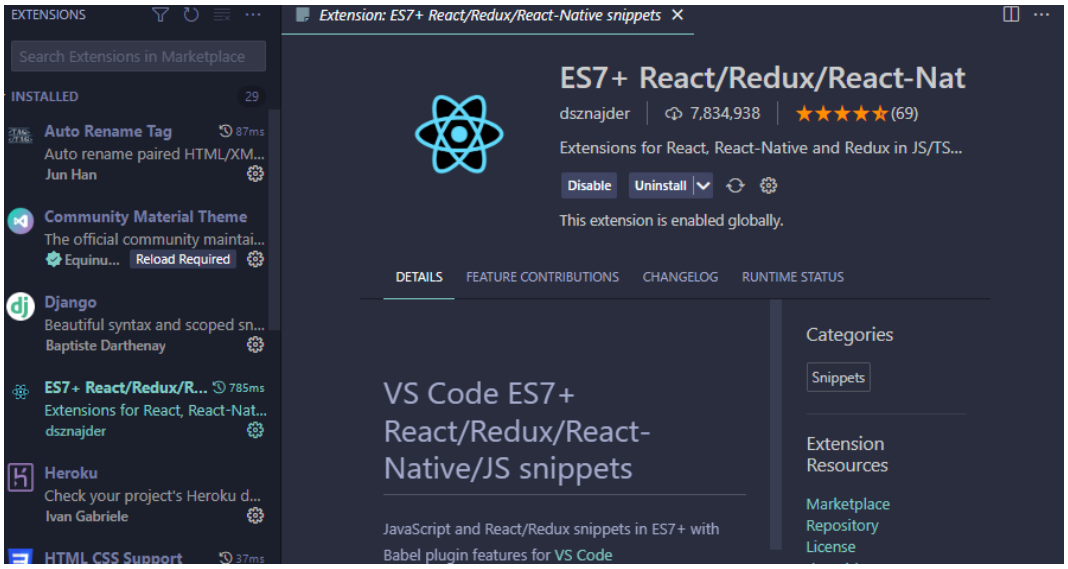


Figure 9. VS Code application

4.2 React Setup.

To set up a new React project, first it needs to be ensured that Node.js and NPM (Node Package Manager) are installed on the system. If they are not already

installed, they can be downloaded from the official Node.js website. Once Node.js and NPM installed are opened, to navigate to the directory where you want to create a project.

From there, the command below is used to create a new React project with Type-script:

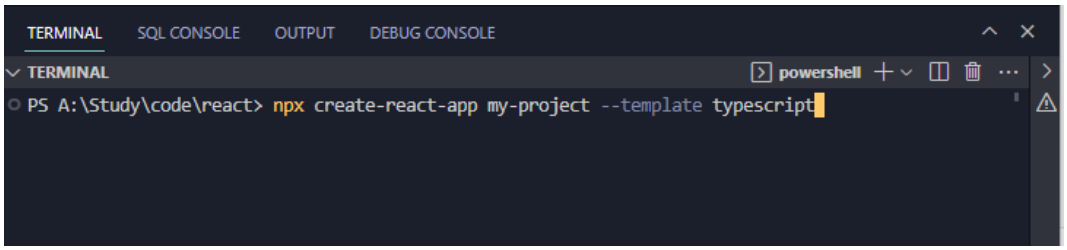


Figure 10. Example for creating new project.

It is advisable to replace "my-project" with the desired name of the project. After running the command, wait for the process to complete, which may take several minutes depending on your internet speed and system resources. Once the com-mand is finished, navigate to the project directory by typing cd my-project in the terminal.

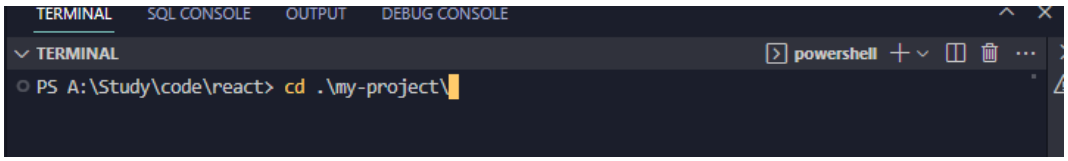


Figure 11. Example for navigating to the project.

Finally, the development server is started by running npm start in the terminal, which will open the project in a web browser.

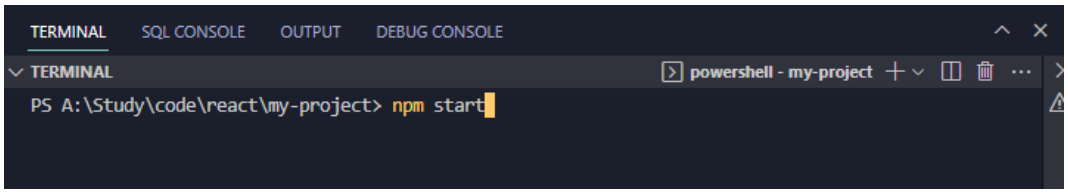


Figure 12. Example for starting the project.

4.3 Version Control

Version control is a critical aspect of any software development project. It provides a systematic way of managing changes to code and other project assets, allowing developers to track changes over time, collaborate effectively, and roll back changes when necessary. In this section, we will discuss the version control system we will use for our project and the best practices that we will follow to ensure efficient and effective collaboration among the development team (Mijacobs , Liz-Casey & EdKaim, 2022).

For our project, we will use Git as our version control system. Git is a widely used distributed version control system that provides powerful branching and merging capabilities, making it an ideal choice for collaborative software development. Additionally, Git integrates seamlessly with many other tools and services that we will use throughout the development process, such as GitHub, a popular web-based hosting service for Git repositories.

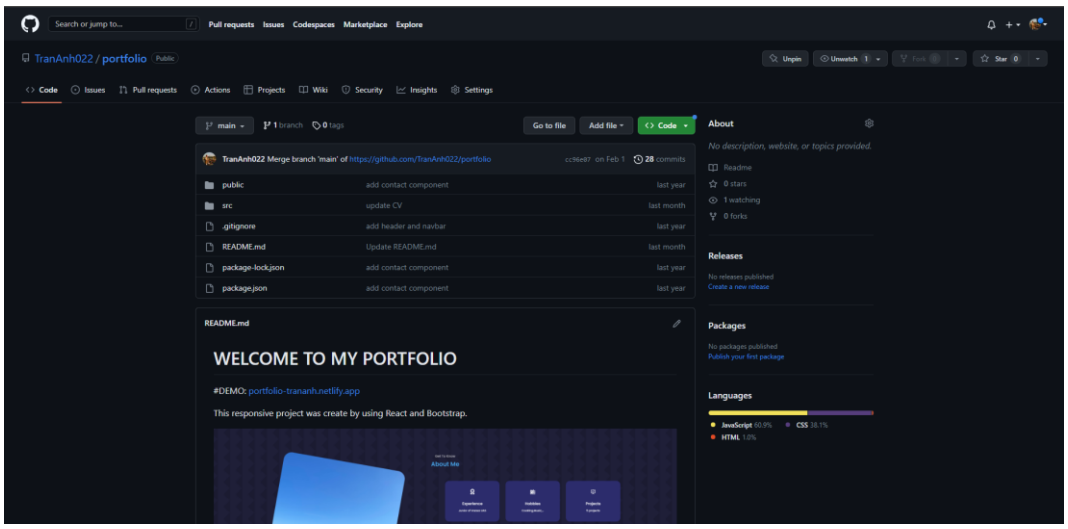


Figure 13. Example of GitHub repository

4.4 Project structure

There is a parent repository that includes multiple small repositories.

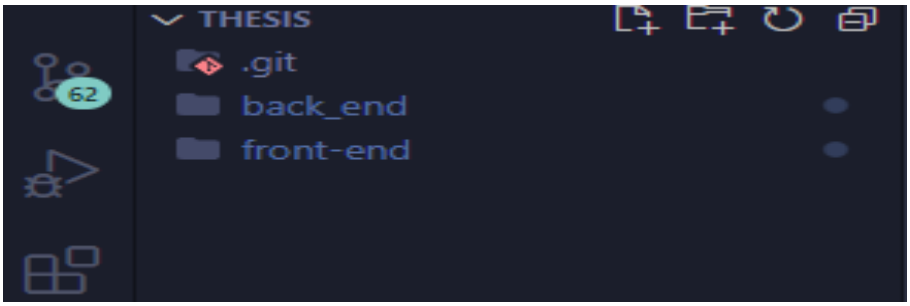


Figure 14. Project Structure

In any project, the *node_modules* directory would contain all the dependencies required for the MERN stack to work. These dependencies can be installed using a package manager such as npm (Node Package Manager) or yarn. When running npm install or yarn install in the project directory, the package manager will automatically download and install all the required dependencies and create the node_modules directory.

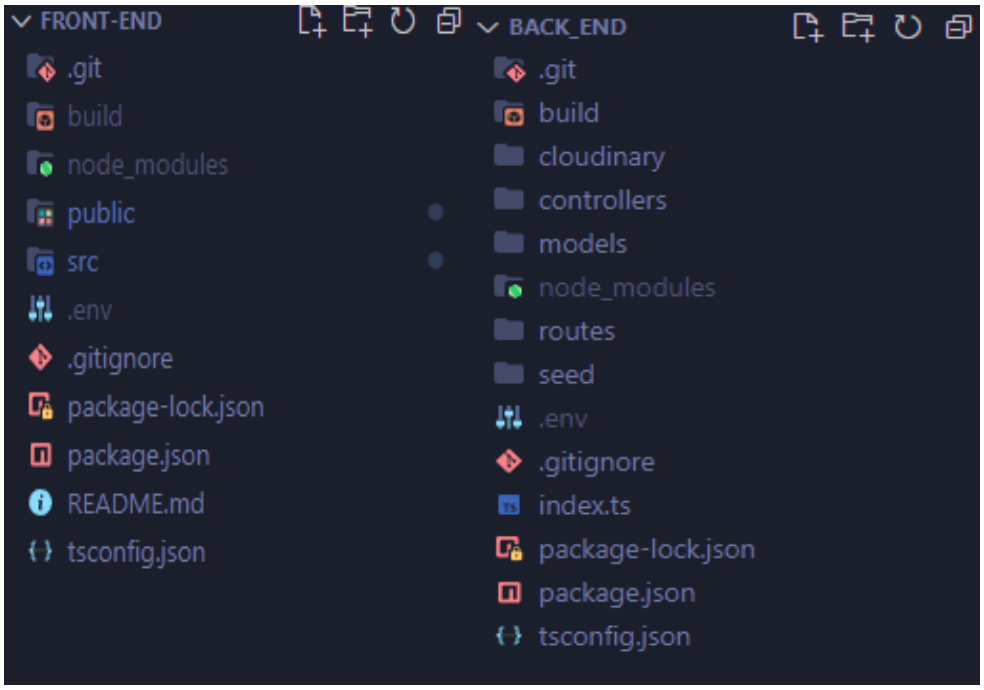


Figure 15. Structure of backend and frontend

4.5 Backend Logic

To run the application, it is necessary to have a server that can handle the processing of data from the database and authentication of login information.

4.5.1 REST API endpoint

In order to create an API endpoint that allows clients to send requests to the server, a router needs to be created using the Express library. The router will help to define the URL paths for API endpoints and the HTTP methods (GET, POST, PUT, DELETE, etc.) that each endpoint will respond to

Table 2. API endpoint for user

Route	Purpose

POST /register	Register the new user
POST /login	Authorization user and get user
POST /password	Send code to email to reset password
POST /password/reset	Compare the code and change new password
GET /logout	Logout user

In addition to the API endpoint for user-related requests, there are also API endpoints available for campsites and reviews within the application. These endpoints allow for the manipulation and retrieval of information related to campsites and their associated reviews, respectively. By using these API endpoints, users of the application can create, read, update, and delete campsite and review data as needed.

Table 3. API endpoint for campsites and reviews

GET /campsites	Get all the campsites
POST /campsites	Create a new campsite
GET /campsites/:id	Get information for single campsite
PUT /campsites/:id	Edit/Update the campsite
DELETE / campsites/:id	Delete the campsite
POST /campsite/:id/reviews	GET all reviews of the campsite
DELETE /campsite/:id/reviews/:reviewId	Delete specific review

In order to organize the code and make it more manageable, all API endpoints are stored in the "routes" folder, which is divided into three files: userRoutes, reviewRoutes, and campingSiteRoutes. This allows for a clear separation of concerns and makes it easier to maintain and modify the code. The userRoutes file handles

all requests related to user authentication and account management, such as login, signup, and logout. The reviewRoutes file handles requests related to reviews, such as creating and deleting reviews. The campingSiteRoutes file handles requests related to campgrounds, such as creating and deleting campgrounds, as well as retrieving and updating campground information. By dividing the endpoints into these separate files, the code is more modular and easier to understand, making it simpler to add new features or modify existing ones in the future.

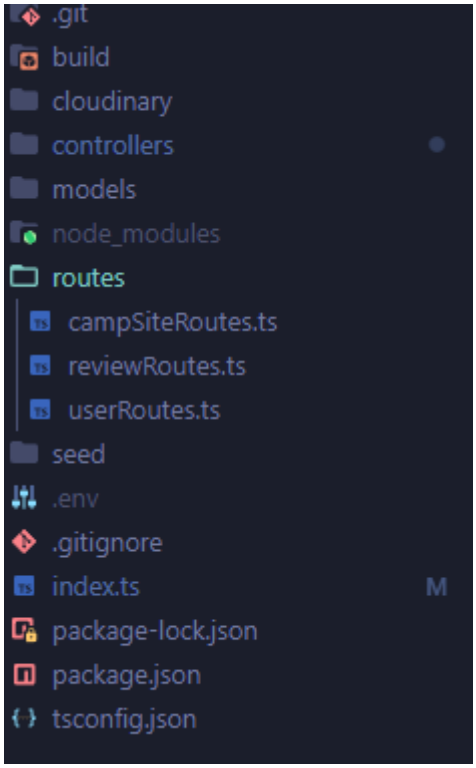


Figure 16. Routes structure

4.5.2 Database Connection

To implement the database, the Mongoose library was used to define the data models and create the necessary schema. Mongoose is an Object-Document Mapping (ODM) library that provides a straightforward way to interact with MongoDB from Node.js. The data was stored in MongoDB as a JSON-like document, which is

flexible and can have varying structure. The first step to create a database is sign up for a MongoDB account, then a cluster, which contains the database is created. After that, store the cluster URL to the environment variable file.

```
const dbUrl = process.env.MONGO_URL || "mongodb://localhost:27017/campsite";
mongoose.connect(dbUrl, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const db = mongoose.connection;
db.on("error", console.error.bind(console, "connection error:"));
db.once("open", () => {
  console.log("Database connected");
});
```

Figure 17. Mongoose connection

Figure 17 above shows the connection is required a URL from environment file or the localhost server. Moreover, the code sets up event listeners on the “db” object to handle the connection events. When an error occurs during the connection process, the code logs the error message to the console using console.error(). When the connection is established successfully, the code logs a message to the console indicating that the database is connected. After connecting to the database, models needed to be created for saving data to database.

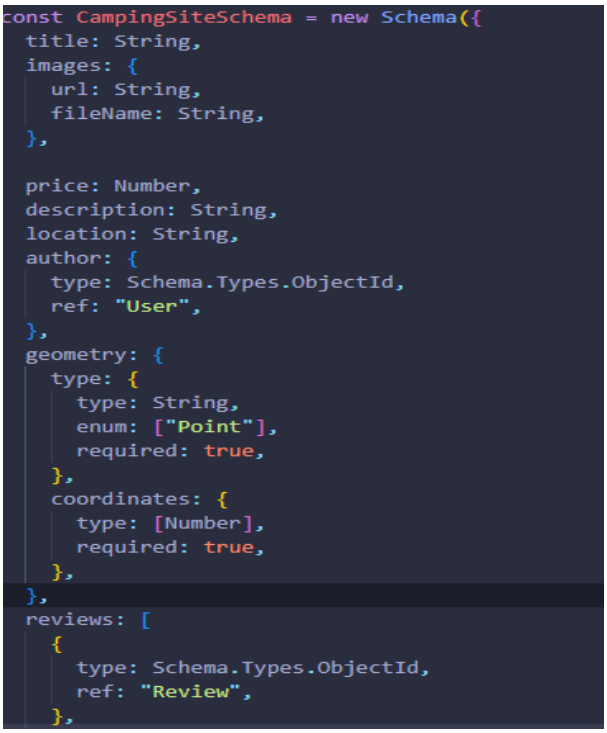


Figure 18. Camping site Schema

Figure 18 shows a basic schema model which includes fundamental information about the camping site. In CampingSiteSchema, by specifying the data type and whether a field is required or not, we can ensure that our data is stored in a consistent and predictable format. We can also perform validation on our data to ensure that it meets certain criteria, such as minimum or maximum length, allowed values, or data range.

When the new camping site was pushed to MongoDB, it will automatically generate a "_id," as shown in Figure 19 below.

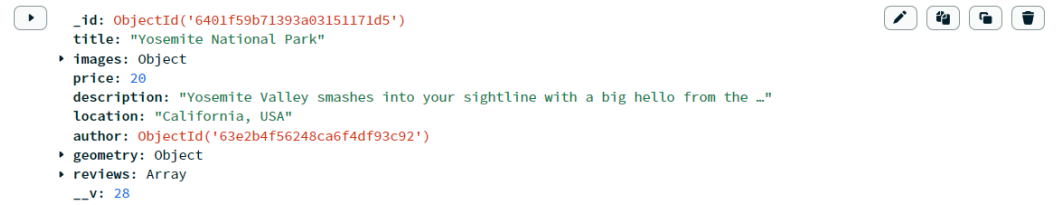


Figure 19. Camping site object in MongoDB.

4.5.3 Mapbox

Mapbox is a powerful location data platform that provides developers with the ability to integrate interactive maps and geospatial functionality into their applications. In this project, Mapbox was used to store and display the geographical data of campsites, including their latitude and longitude coordinates.

The implementation of Mapbox in this project involved creating a custom database schema that was optimized for storing and querying spatial data. By using Mapbox, we were able to create a more intuitive and visually appealing user interface, as well as improve the overall user experience of the application.

```
import mbxGeocoding from "@mapbox/mapbox-sdk/services/geocoding";

const mapBoxToken = process.env.MAPBOX_TOKEN as string;

const geocoder = mbxGeocoding({ accessToken: mapBoxToken as string });
```

Figure 20. How to import the Mapbox in the project.

```
const geoData = await geocoder
  .forwardGeocode({
    query: req?.body?.location,
    limit: 1,
  })
  .send();
```

Figure 21. Mapbox feature (create a geometry point)

Figures 20 and 21 demonstrate how Mapbox converts a location to its corresponding geographic coordinates, which can then be used to create a new campground in our application.

4.5.4 Controller

The controller file is responsible for handling incoming requests and sending responses back to the client. In this project, we have three controller files userController, reviewController, and campingController - each corresponding to a different endpoint. We will go through some significant algorithms or operations that are used in this controller.

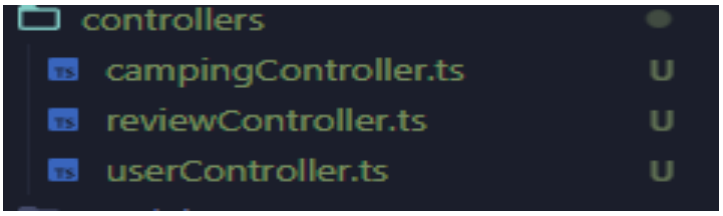


Figure 22. Controller structure

The userController handles all the user-related API requests, including user registration, login, and reset password. It communicates with the user model to store and retrieve user data from the database. The overall logic for user file in Controller is to interact with the user model to perform CRUD operations and return appropriate HTTP responses.

In registration function, When the client sends data to the server via the req.body, the controller retrieves and processes this data. It then utilizes the passport middleware to authenticate and validate the user's credentials, allowing them to successfully register or log in. The controller sends a response status back to the client to inform them of the success or failure of their action.

```
export const createUser = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    const { username, email, password } = req.body;
    const user = new User({ email, username });
    const registerUser = await User.register(user, password); //using the register method from passport middleware
    req.login(registerUser, (err) => {
      if (err) return next(err);
      res.status(200).json({ registerUser });
    });
  } catch (e: any) {
    res.status(400).json({ message: e.message });
  }
};
```

Figure 23. Controller for create new user.

The login and logout functions in the userController rely on the passport middleware for user authentication. When a user logs in, passport validates their credentials against the stored user data. Similarly, when a user logs out, passport handles the session and prevents it from being deleted. This ensures that the user's session remains intact, allowing for a smooth and secure user experience.

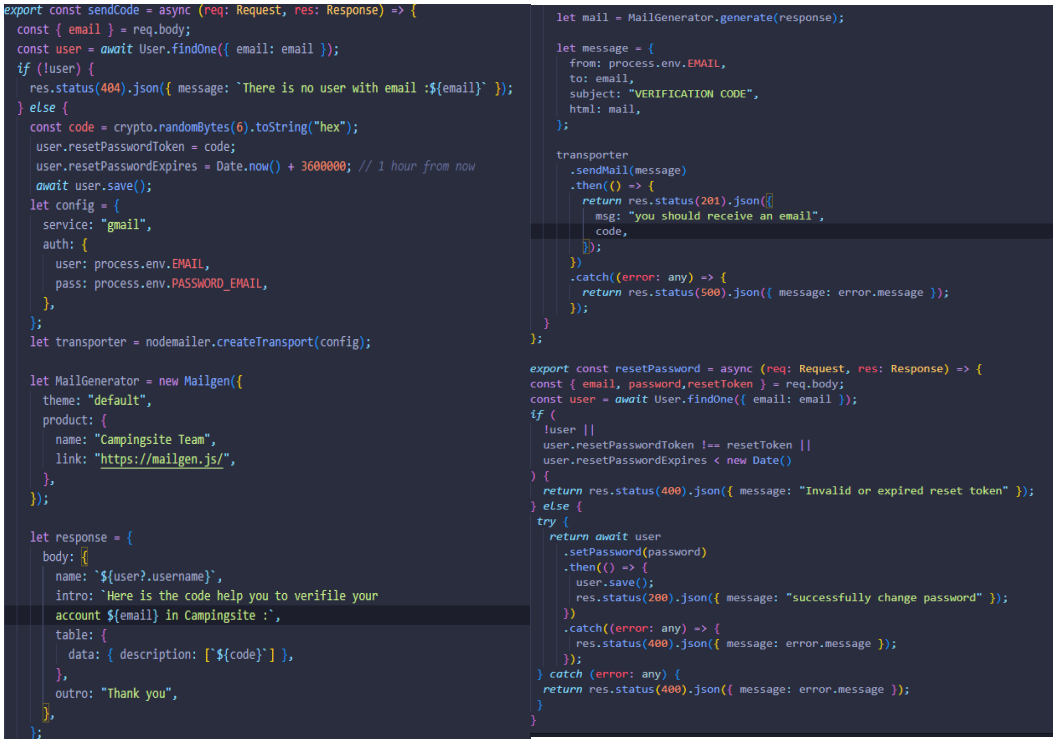
```
// --login--
export const login = async (req: Request, res: Response) => {
  try {
    const { username } = req.body;
    const user = await User.findOne({ username: username });
    if (!user) res.status(400).json(user);
    res.status(200).json({ user });
  } catch (e: any) {
    res.status(500).json({ error: e.message });
  }
};

//--logout--
export const logout = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  req.logout(function (err) {
    if (err) return next(err);
  });
  res.status(200).json({message: "logout"})
};
```

Figure 24. userController for login and logout function

The userController also includes a function to reset the password for users who have forgotten their password. The function searches for the user's email in the database and sends a verification code to their email. Once the user enters the

correct verification code, they are allowed to change their password, and the new password is then saved in the database. This process is implemented using a combination of email sending libraries and database queries to ensure secure and efficient password recovery.



```
export const sendCode = async (req: Request, res: Response) => {
  const { email } = req.body;
  const user = await User.findOne({ email: email });
  if (!user) {
    res.status(404).json({ message: `There is no user with email :${email}` });
  } else {
    const code = crypto.randomBytes(6).toString("hex");
     user.resetPasswordToken = code;
     user.resetPasswordExpires = Date.now() + 3600000; // 1 hour from now
     await user.save();
    let config = {
      service: "gmail",
      auth: {
        user: process.env.EMAIL,
        pass: process.env.PASSWORD_EMAIL,
      },
    };
    let transporter = nodemailer.createTransport(config);

    let MailGenerator = new Mailgen({
      theme: "default",
      product: {
        name: "Campingsite Team",
        link: "https://mailgen.js/",
      },
    });

    let response = {
      body: {
        name: `${user?.username}`,
        intro: `Here is the code help you to verifile your
        account ${email} in Campingsite :`,
        table: {
          data: { description: [`${code}`] },
        },
        outro: "Thank you",
      },
    };
    let mail = MailGenerator.generate(response);

    let message = {
      from: process.env.EMAIL,
      to: email,
      subject: "VERIFICATION CODE",
      html: mail,
    };

    transporter
      .sendMail(message)
      .then(() => {
        return res.status(201).json({
          msg: "you should receive an email",
          code,
        });
      })
      .catch((error: any) => {
        return res.status(500).json({ message: error.message });
      });
  }
};

export const resetPassword = async (req: Request, res: Response) => {
const { email, password,resetToken } = req.body;
const user = await User.findOne({ email: email });
if (
  !user ||
  user.resetPasswordToken !== resetToken ||
  user.resetPasswordExpires < new Date()
) {
  return res.status(400).json({ message: "Invalid or expired reset token" });
} else {
 try {
   return await user
     .setPassword(password)
     .then(() => {
       user.save();
       res.status(200).json({ message: "successfully change password" });
     })
     .catch((error: any) => {
       res.status(400).json({ message: error.message });
     });
 } catch (error: any) {
  return res.status(400).json({ message: error.message });
 }
}
```

Figure 25. userController for reset the password.

Now we move to campingController which is responsible for handling all of the logic related to the management of campsite data, including creating new campsites, editing existing campsites, and deleting campsites. It also handles the retrieval of campsite data to display on the frontend. There are several other functions and associated logic implemented in the campingController that will be explained below.

The getAllCampsite function retrieves all campsites in the database by calling the find method on the Campsite model. It then sends the retrieved data to the client as a JSON response with a status code of 200.

```
export const getAllCampSite = async (_req: Request, res: Response) => {
  try {
    const campSites = await CampSite.find();
    res.status(200).json(campSites);
  } catch (error: any) {
    res.status(500).json({ message: error.message });
  }
};
```

Figure 26. campingController for getting all camping sites.

The getCampSite function retrieves a specific camping site from the database by its Id. It takes the Id from the request parameter and searches for a matching campsite in the database using the findById method on the Campsite model. If a match is found, it sends the campsite data to the client as a JSON response with a status code of 200. If no match is found, it sends an error response with a status code of 404.

```
export const getCampSite = async (req: Request, res: Response) => {
  try {
    const campsite = await CampSite.findById(req.params.id)
      .populate({
        //this is a nested populate. We populate all the review from t
        path: "reviews", // there are many reviews which were written
        populate: {
          path: "author",
        },
      })
      .populate("author");
    if (!campsite) {
      res.status(404).json({ message: "Camping place not found" });
      return;
    }
    res.status(200).json(campsite);
  } catch (error: any) {
    res.status(500).json({ message: error.message });
  }
};
```

Figure 27. campingController use for getting specific camping site.

The next function in the campingController is createCampSite. This function is responsible for creating a new campsite in the database based on the information provided by the client.

```
export const createCampSite = async (req: Request, res: Response) => {
  try {
    const geoData = await geocoder
      .forwardGeocode({
        query: req?.body?.location,
        limit: 1,
      })
      .send();
     const userTypeWithId = req.user as UserType & { _id: string };
    const { title, location, price, description } = req.body;

    const campsite = new CampSite({
      title,
      images: {
        url: req?.file?.path,
        fileName: req?.file?.fieldname,
      },
      description: description,
      author: userTypeWithId?._id,
      location,
      price,
      geometry: geoData.body.features[0].geometry,
    });
    await campsite.save();
    res.status(200).json(campsite);

  } catch (error) {
    (error: any) => {
      res.status(500).json({ message: error.message });
    };
  }
};

//--Edit campsite--
```

Figure 28. campingController for creating new camping place.

The createCampSite function, as shown in Figure 28, utilizes the Campsite model to create a new campsite in the database, using the information provided by the client in the request body. Additionally, the function uses Mapbox to generate the geographic coordinates for the camping place, which are then added to the database as the geometry point for the campsite.

The editCampsite and deleteCampsite functions are crucial for managing campsite data. These functions utilize the ID provided by the client in the request body to search for the campsite that needs to be updated or deleted in the database. The editCampsite function is responsible for updating the campsite data with the new information provided by the client, while the deleteCampsite function is responsible for deleting the campsite from the database The implementation of these functions can be seen in Figure 29 below.

```
export const editCampsite = async (req: Request, res: Response) => {
  try {
    const { id } = req.params;
    let image;
    const campsite = await CampSite.findById(id);
    if (req?.file === undefined) {
      if (!req.body?.imageUrl) {
        image = { url: "", fileName: "" };
      } else {
        image = {
          url: campsite.images.url,
          fileName: campsite.images.fileName,
        };
      }
    } else {
      image = {
        url: req.file?.path,
        fileName: req.file?.fieldname,
      };
    }

    const geoData = await geocoder
      .forwardGeocode({
        query: req.body.location,
        limit: 1,
      })
      .send();
    if (image?.fileName !== campsite.images.fileName) {
      await cloudinary?.uploader?.destroy(campsite.images.fileName);
    }
    const campsiteUpdated = await CampSite.findByIdAndUpdate(
      id,
      {
        ...req.body,
        images: req.file === undefined ? campsite.images : image,
        geometry: geoData.body.features[0].geometry,
      },
      { new: true }
    );
    res.status(200).json(campsiteUpdated);
  } catch (error: any) {
    res.status(500).json({ message: error.message });
  }
};

export const deleteCampsite = async (req: Request, res: Response) => {
  try {
    const { id } = req.params;
    await CampSite.findByIdAndDelete(id);
    res.status(200).json("deleted successfully!!!");
  } catch (error: any) {
    res.status(500).json({ message: error.message });
  }
};
```

Figure 29. campingController for editing and deleting.

The reviewController file in the controllers folder contains two important functions that allow clients to create and delete their reviews. These functions are responsible for managing the review data in the database, as shown in Figure 30 below.

```
export const createReview = async (req: Request, res: Response) => {
  try {
    const { id } = req.params;
    const campsite = await CampSite.findById(id);
    const review = new Review({
      body: req.body.comment,
      rating: req.body.rating,
    });
    const userTypeWithId = req.user as UserType & { _id: string };
    review.author = userTypeWithId?._id;
    campsite.reviews.push(review);
    await review.save();
    await campsite.save();
    res.status(200).json({
      message: "Congratulations on successfully creating a new review!!!",
      review,
    });
  } catch (e: any) {
    res.status(500).json(e.message);
  }
};

export const deleteReview = async (req: Request, res: Response) => {
  try {
    const { id, reviewId } = req.params;
    await CampSite.findByIdAndUpdate(id, {
      $pull: { reviews: reviewId },
    }); //we using pull method from mongoose to remove(take of) the review
    await Review.findByIdAndDelete(reviewId); // this method will delete the
    res.status(200).json({
      message: "Delelte the comment successfully",
    });
  } catch (e: any) {
    res.status(500).json(e.message);
  }
};
```

Figure 30. reviewController for creating and deleting review.

4.6 Front End Logic

The frontend of the application is responsible for rendering the user interface and handling user interactions. It communicates with the backend through HTTP requests to exchange data and perform various operations, such as creating and

deleting campsites, managing reviews, and handling user interactions. We will go through each section and explain some important code in the front-end.

4.6.1 Login and Register

The front-end of the application includes features for user login and register, which allow users to register, login, and manage their accounts. The registration form collects user input for username, email, and password, and submits the data to the backend API for user creation. The login form allows users to enter their credentials, which are then authenticated against the backend API. Once logged in, users can access restricted areas of the application based on their role such as create leave a comment or create a new camping place.

```
const [password, setPassword] = useState("");
const [username, setUserName] = useState("");
```

```
const loggedInResponse = await fetch(
  `${process.env.REACT_APP_BASE_URL}/login`,
  {
    method: "POST",
    credentials: "include",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ username, password }),
  }
```

Figure 31. Login function

```
const [password, setPassword] = useState("");
const [username, setUsername] = useState("");
const [email, setEmail] = useState("");
```

```
const registerResponse = await fetch(
  `${process.env.REACT_APP_BASE_URL}/register`,
  {
    method: "POST",
    credentials: "include",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ username, password, email }),
  }
```

Figure 32. Function for registration

4.6.2 Home Page

The home page is the landing page of the application, where users can explore campsites and interact with the map-based interface. The home page includes various features, such as a map component to display the campsites with markers, and a list of campsites with basic information and ratings. Users can click on the markers or campsite cards to view detailed information about a particular campsite, including reviews and ratings.

```
const getCamsites = async () => {
  const response = await fetch(
    `${process.env.REACT_APP_BASE_URL}/campsites`,
    {
      credentials: "include",
      method: "GET",
    }
  );
  const data = await response.json();
  dispatch(setCampsites({ campsites: data }));
};
```

Figure 33. Function for retrieving all campsites from database.

4.6.3 Campsite Page

The campsite page displays detailed information about the campsite, such as its name, location, description, amenities, and photos. This information is retrieved from the backend API and rendered in a user-friendly format for users to easily understand and evaluate the campsite. Besides, users can view reviews and ratings submitted by other users for the campsite. The reviews are displayed in a list format, showing the reviewer's name, rating, and comments. Users can read the reviews to gain insights and make informed decisions about the campsite.

```
const getCampsite = async () => {
  const response = await fetch(
    `${process.env.REACT_APP_BASE_URL}/campsites/${id}`,
    {
      method: "GET",
      credentials: "include",
      headers: {
        "Content-Type": "application/json",
      },
    }
  );
  const responsed = await response.json();
  dispatch(showCampsite({ campsite: responsed }));
};
```

Figure 34. Function for retrieving specific campsite.

4.6.4 Create and Edit Campsite Page

The create and edit campsite page allows users to add new campsites or modify existing campsite information. It includes a form for users to enter or update campsite details, such as name, location, description, and upload photos. The form also includes validation and error handling mechanisms for data submission. The page communicates with the backend API to send and receive data and uses Cloudinary for uploading and storing photos. One important feature of the create and edit campsite page is the ability for users to upload photos of the campsite. This is facilitated using Cloudinary, a cloud-based media management platform. Users can select and upload photos from their local device, which are then sent to Cloudinary for storage. Cloudinary provides features such as image optimization, transformation, and hosting, which enhance the performance and user experience of the application by delivering optimized images to the frontend.

```
import * as Yup from "yup";

const schema = Yup.object().shape({
  title: Yup.string().required("Title is required"),
  price: Yup.number()
    .required("Price is required")
    .min(0, "Price must be greater than or equal to 0"),
  location: Yup.string().required("Location is required"),
  description: Yup.string().required("Description is required"),
});

export default schema;
```

Figure 35. Form Validation

```
const [camp, setCamp] = useState({
  title: "",
  location: "",
  price: 0,
  description: "",
});
```

```
const createCamp = async (e: { preventDefault: () => void }) => {
  e.preventDefault();
  const formData = new FormData();
  formData.append("title", camp.title);
  formData.append("location", camp.location);
  formData.append("price", String(camp.price));
  formData.append("description", camp.description);
  formData.append("image", img as File);

  schema
    .validate(Object.fromEntries(formData))
    .then(async (validate: FormInput) => {
      try {
        await axios
          .post(`${process.env.REACT_APP_BASE_URL}/campsites`, validate, {
            withCredentials: true, // Send cookies with request
            headers: {
              "Content-Type": "multipart/form-data", // Set appropriate headers
            },
          })
```

Figure 36. Function for creating new campsite.

5 OUTCOME OF THE PROJECT

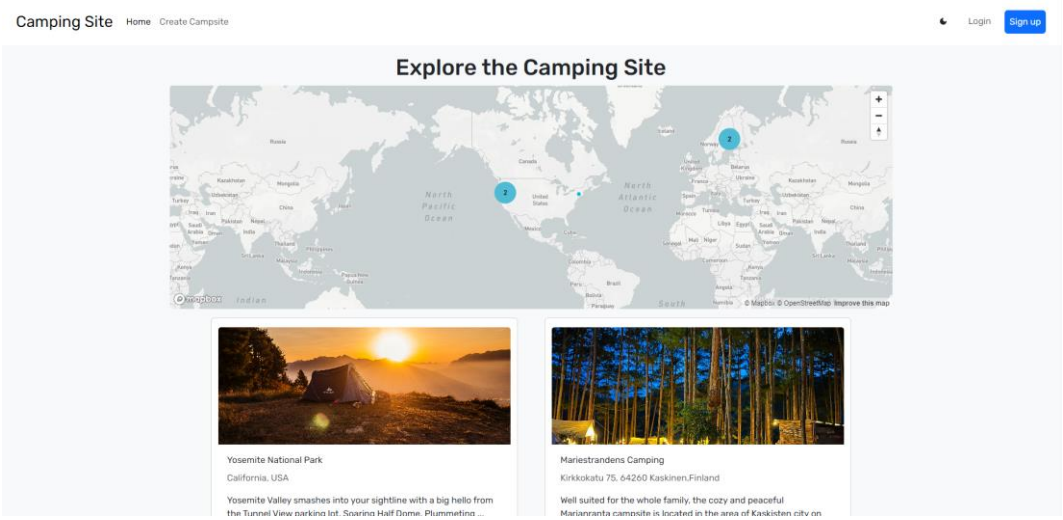


Figure 37. Homepage light mode

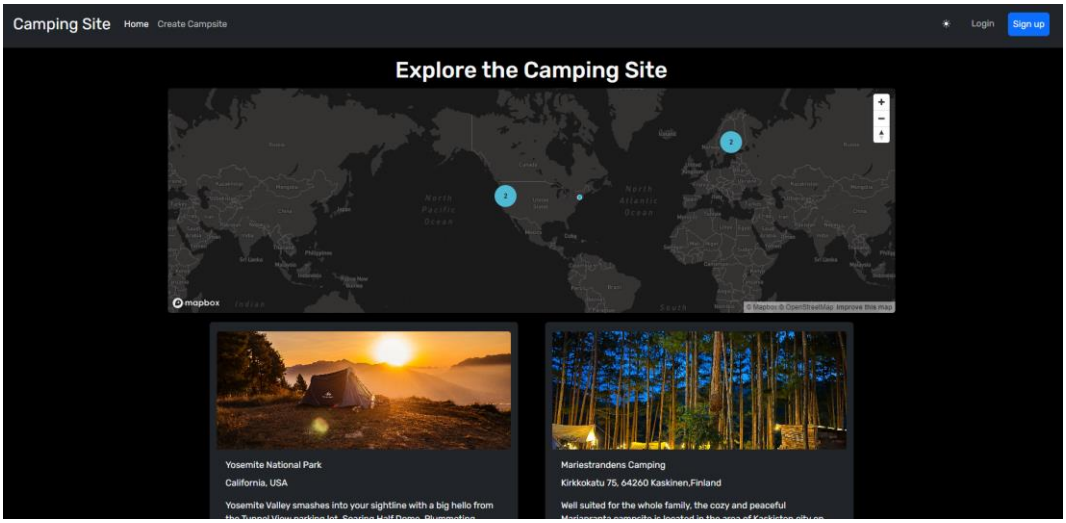


Figure 38. Homepage dark mode

The homepage of the camping site will feature a responsive navbar in the header that provides easy navigation for users to different sections of the website. The navbar may include links to Home, createCampSite , Login, and Register pages, allowing users to access different parts of the website efficiently. One of the unique features of the homepage is the availability of a dark mode. Users will have the option to switch between light and dark mode, depending on preference.

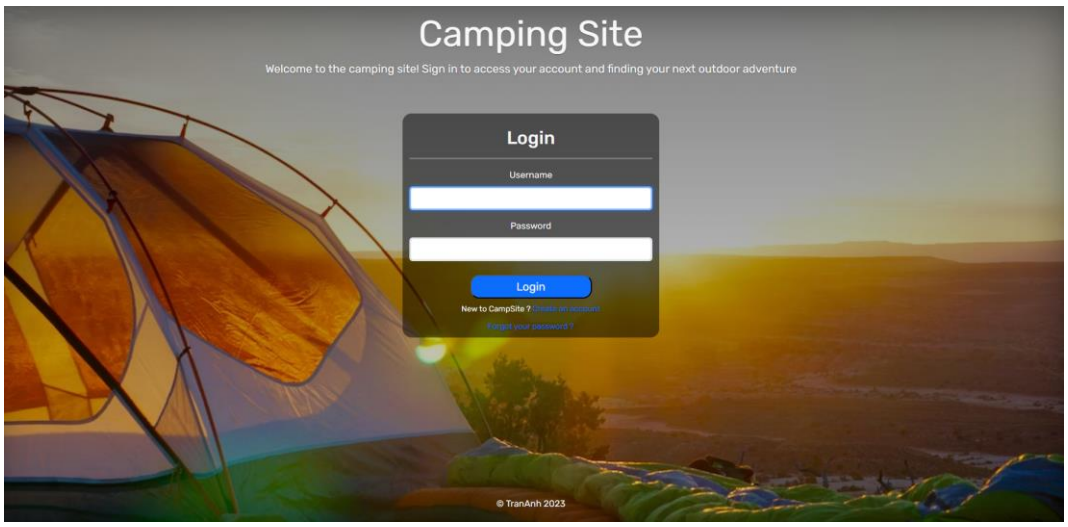


Figure 39. Login Page.

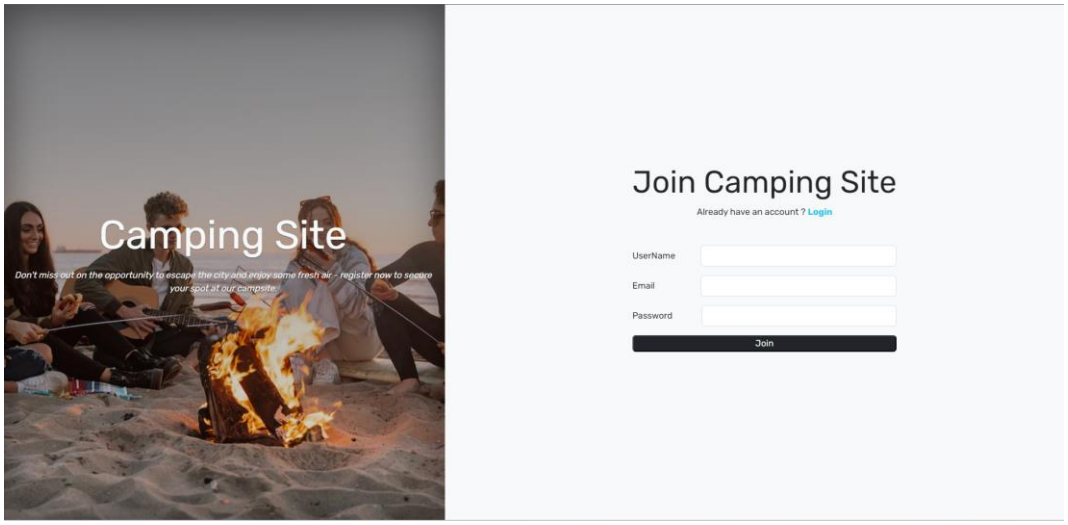


Figure 40. Register Page

The login and register page will have a secure and user-friendly design. The login page will prompt users to enter their credentials, such as email and password, and will include password recovery options. The register page will have a registration form that collects necessary information from users, such as name, email, password for registration. It will also include validation checks and error handling to ensure accurate registration.

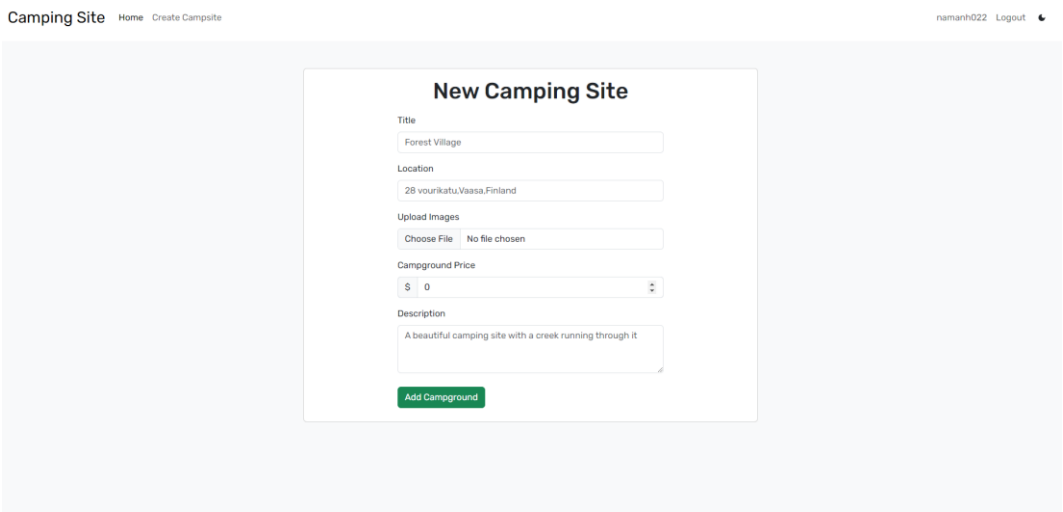


Figure 41. Create campsite page.

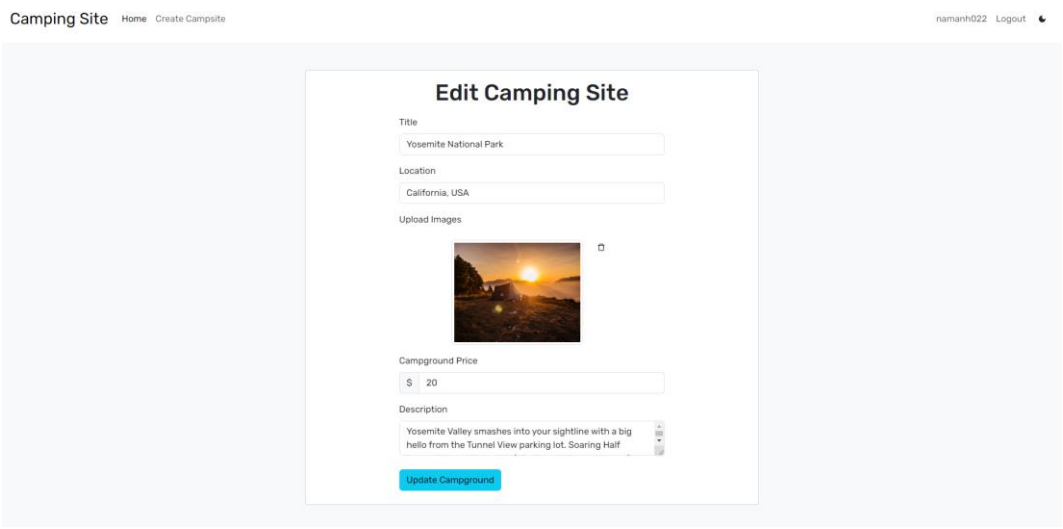


Figure 42. Edit campsite page.

The create and edit page will allow users to contribute to the website by sharing their experiences about campgrounds. The create page will have a form that allows users to input campground information, such as campground name, location, price, and a review or description. It will also allow users to upload relevant images to enhance their campground submission. The edit page will allow users to edit or update their previously submitted campground information, providing them with the ability to make changes as needed. Both the create and edit pages will have

validation checks and error handling to ensure accurate data input and prevent misuse.

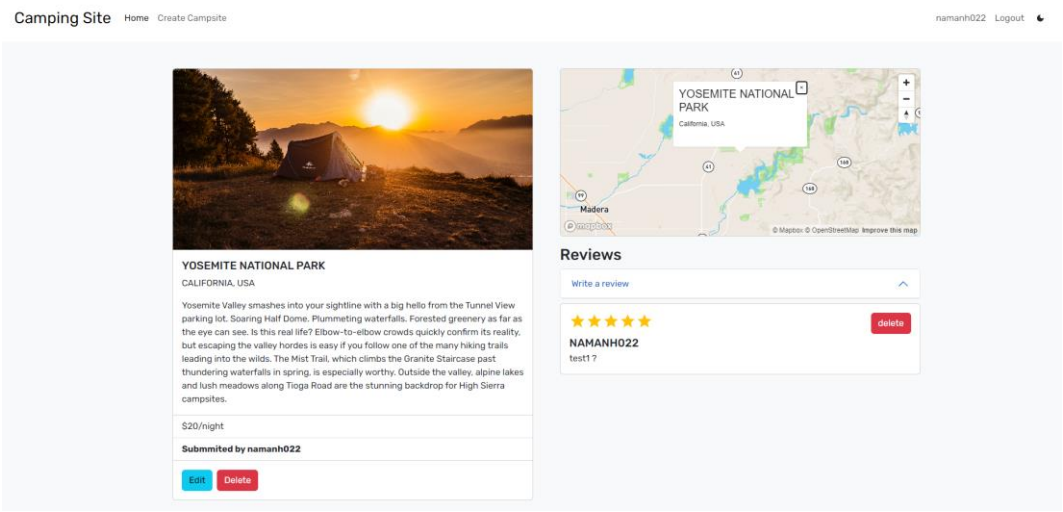


Figure 43. Specific campsite page

The specific campsite page is a dedicated page that provides detailed information about a specific campground listed on the website. When users click on a campground listing from the homepage or search results, they will be directed to the specific campsite page. It also displays reviews and ratings from other users who have visited the campground. Users can read and leave reviews, providing valuable insights and feedback for other potential campers.

Additionally, the specific campsite page may include a map that shows the exact location of the campsite. Users can zoom in, zoom out, and pan the map to explore the surrounding area and get directions to the campsite.

6 CONCLUSIONS

In conclusion, this thesis has presented a comprehensive overview of the development of a camping website prototype, covering both the backend and frontend aspects. The backend is built using Node.js, Express, and MongoDB, and includes important functionalities such as user authentication and authorization, campsite and review management, and API endpoints for communication with the frontend. The frontend is developed using modern web technologies including React, Redux, and TypeScript, and includes features, such as form validation, location point for campsite, and a user-friendly interface. The prototype successfully meets the initial goal of providing a platform for camping enthusiasts to register as members, share their experiences about campgrounds they have personally visited, and find information about suitable camping locations.

Throughout the project, I have gained a deeper understanding of full-stack web development, including both the backend and frontend aspects. I have learned how to effectively utilize technologies, such as Node.js, Express, MongoDB, React, Redux, and TypeScript to build a robust and user-friendly application.

Additionally, I have acquired knowledge and experience in implementing essential functionalities, such as user authentication and authorization, API development, form validation, and integration with external services. These skills will undoubtedly be valuable in future projects and endeavors.

One of the key takeaways from this project is the importance of proper planning and architecture design. By following the MVC design pattern and organizing the codebase into modular components, the development process was more efficient and maintainable. I have also learned the significance of using version control systems such as Git, which enabled seamless collaboration and easy tracking of changes throughout the development cycle.

If I were to do the project all over again, I would still choose the MERN stack because it allowed for easy maintenance by utilizing JavaScript throughout the entire stack. Additionally, I would still prefer using a NoSQL database like MongoDB for its flexibility and scalability in managing campsite data, user reviews, and ratings. However, I would make a slight change to the user authentication process by using JSON Web Tokens (JWT) instead of relying on the Passport middleware. This would allow for greater flexibility and easier integration with other systems.

The implementation of the camping site project has achieved the initial goal of providing a platform for users to find and share information about campgrounds. However, there are several things that would need to be considered if this application were to be published. The addition of a search bar would greatly improve the user experience by allowing users to easily search for a specific campground or location. Furthermore, cybersecurity to ensure the confidential information of the user would be required.. Lastly, a mobile application could be developed to increase accessibility and convenience for users who prefer to use their smartphones or tablets.

REFERENCES

Astari , S. (2023). What Is HTML? Hypertext Markup Language Basics for Beginners. Retrieved from: https://www.hostinger.com/tutorials/what-is-html. Accessed 28.3.2023.

Davidson, T. (2023). Is Bootstrap a Framework? | Clean Commit. [online]. Retrieved from: https://cleancommit.io/blog/is-bootstrap-a-framework/.Accessed 6.4.2023.

Gomes, M. (2023). TypeScript: An Introduction to the Statically Typed Superset of JavaScript - DEV Community. Retrieved from: https://dev.to/matheusgomes062/typescript-an-introduction-to-the-statically-typed-superset-of-javascript-4hgo.Accessed 5.4.2023.

Google Trends .React, angular, Vue - Explore . Retrieved from: https://trends.google.com/trends/explore?date=now%201-d&q=react,angular,vue&hl=en. Accessed 28.3.2023

Kumar Ray, A. (2023). Complete Introduction to ReactJS - Coding Ninjas. Retrieved from: https://www.codingninjas.com/codestudio/library/complete-introduction-to-reactjs. Accessed 28.3.2023.

MongoDB .Why Use MongoDB And When To Use It? Retrieved from: https://www.mongodb.com/why-use-mongodb. Accessed 9.4.2023.

Mdn web docs.CSS: Cascading Style Sheets .Retrieved from:https://developer.mozilla.org/en-US/docs/Web/CSS. Accessed 30.3.2023.

Mijacobs, LizCasey & EdKaim, (2022). What is version control? [online] learn.microsoft.com. Retrieved from: https://learn.microsoft.com/en-us/devops/develop/git/what-is-version-control.Accessed 10.4.2023.

Redux. Getting Started with Redux. [online]. Retrieved from: https://redux.js.org/introduction/getting-started.Accessed 5.4.2023.

Semah, B. (2022). What Exactly is Node.js? Explained for Beginners.freecodecamp.org. Retrieved from: https://www.freecodecamp.org/news/what-is-node-js/.Accessed 6.4.2023.

Vegibit. Introduction to Express.js: Building Web Applications with Node.js. [online]. Retrieved from: https://vegibit.com/introduction-to-express-js-building-web-applications-with-node-js/. Accessed 8.4.2023.

Williams, M. (2022). What Is JavaScript Used For? | ComputerScience.org. Retrieved from: https://www.computerscience.org/bootcamps/guides/javascript-uses/.Accessed 1.4. 2023.

Zola, A. (n.d). What is a Bootstrap and how does it work?. Retrieved from: https://www.techtarget.com/whatis/definition/bootstrap.Accessed 6.4.2023.