Web Development Using React.js

Layth Al-Zamili

Master's thesis
2024

Tomas Bata University in Zlín
Faculty of Applied Informatics

Tomas Bata University in Zlín
Faculty of Applied Informatics
Department of Informatics and Artificial Intelligence

Academic year: 2023/2024

ASSIGNMENT OF DIPLOMA THESIS

(project, art work, art performance)

Name and surname:	**Layth Salah Yahyah Al-Zamili**
Personal number:	**A20672**
Study programme:	**N0613A140023 Information Technologies**
Specialization:	**Software Engineering**
Type of Study:	**Full-time**
Work topic:	**Web Development Using React.js**
Work topic in English:	**Web Development Using React.js**

Theses guidelines

1. Clearly state the project's purpose and why the MERN stack was chosen.
2. Provide a brief overview of the MERN stack, highlighting the role of each technology.
3. Outline the development process, emphasizing key methodologies employed.
4. Build a web app and briefly explain the primary app features.
5. Suggest future improvements.

Form processing of diploma thesis: **printed/electronic**
Language of elaboration: **English**

Recommended resources:

1. Stefanov, Stoyan. React: Up and Running: Building Web Applications. 2016.
2. Banks, Alex, and Eve Porcello. Learning React: Functional Web Development With React and Redux. 2017.
3. Casciaro, Mario, and Luciano Mammino. Node.js Design Patterns. Packt Publishing Ltd, 2016.
4. Mardan, Azat. Express.js Guide: The Comprehensive Book on Express.js. Azat Mardan, 2014.
5. Chodorow, Kristina, and Michael Dirolf. MongoDB: The Definitive Guide. "O'Reilly Media, Inc.," 2010.

Supervisors of diploma thesis: **Ing. Bc. Pavel Vařacha, Ph.D.**
Department of Informatics and Artificial Intelligence

Date of assignment of diploma thesis: **November 5, 2023**
Submission deadline of diploma thesis: **May 13, 2024**

doc. Ing. Jiří Vojtěšek, Ph.D. m.p.
Dean

prof. Mgr. Roman Jašek, Ph.D., DBA m.p.
Head of Department

In Zlín January 5, 2024

I hereby declare that:

- I understand that by submitting my Master´s thesis, I agree to the publication of my work according to Law No. 111/1998, Coll., On Universities and on changes and amendments to other acts (e.g. the Universities Act), as amended by subsequent legislation, without regard to the results of the defence of the thesis.
- I understand that my Master´s Thesis will be stored electronically in the university information system and be made available for on-site inspection, and that a copy of the Master´s Thesis will be stored in the Reference Library of the Faculty of Applied Informatics, Tomas Bata University in Zlín, and that a copy shall be deposited with my Supervisor.
- I am aware of the fact that my Master´s Thesis is fully covered by Act No. 121/2000 Coll. On Copyright, and Rights Related to Copyright, as amended by some other laws (e.g. the Copyright Act), as amended by subsequent legislation; and especially, by §35, Para. 3.
- I understand that, according to §60, Para. 1 of the Copyright Act, TBU in Zlín has the right to conclude licensing agreements relating to the use of scholastic work within the full extent of §12, Para. 4, of the Copyright Act.
- I understand that, according to §60, Para. 2, and Para. 3, of the Copyright Act, I may use my work - Master´s Thesis, or grant a license for its use, only if permitted by the licensing agreement concluded between myself and Tomas Bata University in Zlín with a view to the fact that Tomas Bata University in Zlín must be compensated for any reasonable contribution to covering such expenses/costs as invested by them in the creation of the thesis (up until the full actual amount) shall also be a subject of this licensing agreement.
- I understand that, should the elaboration of the Master´s Thesis include the use of software provided by Tomas Bata University in Zlín or other such entities strictly for study and research purposes (i.e. only for non-commercial use), the results of my Master´s Thesis cannot be used for commercial purposes.
- I understand that, if the output of my Master´s Thesis is any software product(s), this/these shall equally be considered as part of the thesis, as well as any source codes, or files from which the project is composed. Not submitting any part of this/these component(s) may be a reason for the non-defence of my thesis.

I herewith declare that:

- I have worked on my thesis alone and duly cited any literature I have used. In the case of the publication of the results of my thesis, I shall be listed as co-author.
- That the submitted version of the thesis and its electronic version uploaded to IS/STAG are both identical.

In Zlín; dated: .....

Student´s Signature

ABSTRAKT

Tato práce využívá React.js k vytvoření plnohodnotné webové aplikace demonstrující uživatelskou autentizaci, tvorbu obsahu, sociální interakci a správu dat. Využívá moderní technologický stack, včetně React, Redux, Node.js a MongoDB, aby poskytla uživatelsky příjemný a responzivní zážitek na různých velikostech obrazovky. Projekt ukazuje účinnost React.js při vývoji složitých webových aplikací a poskytuje vhledy do výzev a jejich řešení.

Klíčová slova: React.js, Webový vývoj, Autentizace, Tvorba obsahu, Sociální interakce, Responzivní design

ABSTRACT

This thesis leverages React.js to build a full-featured web application demonstrating user authentication, content creation, social interaction, and data management. It utilizes a modern tech stack including React, Redux, Node.js, and MongoDB to deliver a user-friendly and responsive experience across various screen sizes. The project showcases the effectiveness of React.js in complex web development and provides insights into challenges and solutions encountered.

Keywords: React.js, Web development, Authentication, Content creation, Social interaction, Responsive design

ACKNOWLEDGEMENTS

I am profoundly grateful to Ing.Bc.Pavel Vařacha, Ph.D. for his indispensable guidance during my master's thesis. His expertise and unwavering support played a pivotal role in the successful completion of my research. I am honored to have had him as my supervisor.

I hereby declare that the print version of my Master's thesis and the electronic version of my thesis deposited in the IS/STAG system are identical.

Contents

INTRODUCTION

Web development is a constantly evolving industry, and with the advent of single-page applications, the demand for efficient and user-friendly web experiences has become more pressing. One of the most popular tools for meeting these demands is React.js, a JavaScript library developed by Facebook in 2013.

The modern web landscape demands interactive and dynamic user experiences. React.js, a popular JavaScript library, has emerged as a powerful tool for building complex and engaging web applications.

This thesis explores the capabilities of React.js through the design and development of a full-featured web application.This application demonstrates a rich set of functionalities, including user authentication, content creation and management, social interaction features, and robust data handling. To achieve this, the project leverages a modern technology stack, incorporating React.js on the frontend and Node.js with MongoDB on the backend.

The following sections will delve deeper into the specific technologies employed, the functionalities implemented within the application, and the research questions addressed by this thesis.

Ultimately, this work aims to showcase the effectiveness of React.js in web development and provide valuable insights for developers seeking to build feature-rich and user-friendly web applications.

I. THEORETICAL PART

1 WEB DEVELOPMENT TECHNOLOGIES NOWADAYS

This chapter will begin by imagining the current state of the web technology market. First, we will discuss front-end technologies, followed by back-end technologies. As of 2023, the state of web development technologies is constantly evolving, with new frameworks, tools, and technologies emerging regularly. Some of the notable trends and developments in web development technologies in recent years include:

- Increased focus on performance and speed: With users' expectations for fast and responsive web experiences, web developers are increasingly prioritizing performance and speed. This has led to the adoption of technologies such as serverless computing, caching, and content delivery networks (CDNs) to help improve website loading times and overall performance.
- Greater adoption of JavaScript frameworks: JavaScript continues to be the dominant language for web development, and JavaScript frameworks such as React, Angular, and Vue.js are widely used by developers for building complex and dynamic web applications.
- Rise of static site generators: Static site generators have gained popularity among developers, offering benefits such as faster load times, better security, and improved scalability. Popular static site generators include Gatsby, Hugo, and Jekyll.
- Adoption of new web standards: New web standards such as WebAssembly, Web Components, and Progressive Web Apps (PWAs) have emerged, offering developers new ways to build fast, secure, and responsive web applications.
- Emphasis on accessibility and inclusivity: Web developers are increasingly focused on making their websites and applications more accessible and inclusive, with an emphasis on designing for all users, including those with disabilities.
- Continued growth of mobile-first development: With the increasing use of mobile devices for browsing the web, web developers are increasingly adopting mobile-first development practices to ensure their websites and applications are optimized for mobile devices.

- Advancements in artificial intelligence and machine learning: With the rise of artificial intelligence and machine learning, web developers are exploring ways to incorporate these technologies into web development, such as using machine learning models for personalization or chatbots for customer support.

Overall, the state of web development technologies is constantly evolving, with new technologies and trends emerging regularly. Developers must stay up to date with these changes to build modern, responsive, and performant web applications.

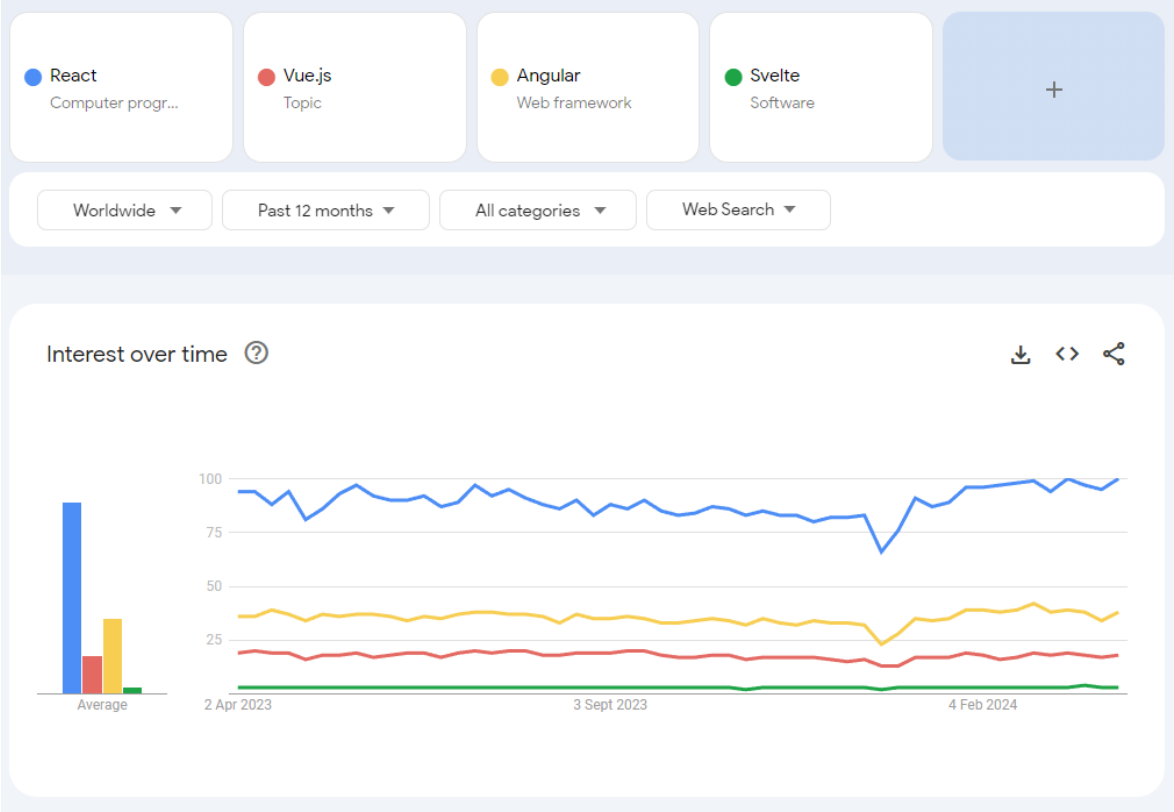

Figure 1 Graph of usage of the front-end framework's technologies [5].

1.1 Front-end technologies

Front-end technologies refer to the technologies and tools used in web development to create user interfaces and enhance the user experience on the client side (i.e., in the browser: buttons, links and more). There are currently a multitude of such applications used to develop diverse frameworks, libraries, and tools. There are some that will be described.

1.1.1 HyperText Markup Language(HTML)

HTML, or Hypertext Markup Language, is a standard markup language used to create the structure and content of web pages. It is the foundation of the web and is supported by all modern web browsers. HTML has been around for a long time. Its roots go back to at least 1980, with Tim Berners-Lee‘s project ENQUIRE. And actually, the concept of hypertext goes back even further than that. The concept first appeared in the early 1940s and was named and demonstrated in the 1960s[6].In 1989, Lee proposed a new hypertext system based on the ideas of ENQUIRE (and other systems, such as Apple’s HyperCard). This became the first version of what we now call HTML [6]. Since then, the language has been in constant development. The specification is managed by the World Wide Web Consortium (Berners-Lee is still the director, as of 2018), and the Web Hypertext Application Technology Working Group. (So, if you don’t like HTML5, these are the people to blame.) [6].The language has evolved over all this time because web development has changed. We do things with web pages and HTML today that were never dreamt of by the early developers and implementers of the language. A web page is no longer just a document; it is likely to be a full-scale web application. And even when it is “just a document,” we want search engines and other tools to understand the content of the website. We aren’t just creating pages for human readers anymore, but for artificially intelligent systems that collect and manipulate information [6].

1.1.1.1 HTML Basic Syntax and page structure

HTML is a text-based language that uses tags to define elements. The most basic element is the <html> tag, which contains the <head> and <body> tags. The <head> tag contains information about the web page, such as the title and meta data, while the <body> tag contains the visible content of the web page. Every proper HTML page should start with a document

type declaration. This declaration tells the browser that this is an HTML document, or what version of HTML it is. Such a declaration of an HTML document is indicated in the next image.

```
<!DOCTYPE html>
<html>
<head>
    <title>My Web Page</title>
</head>
<body>
    <h1>Welcome to my web page</h1>
    <p>This is the content of my web page</p>
</body>
</html>
```

An HTML document with such a fundamental structure is referred to as a boilerplate, and an example of one may be seen in the source code excerpt below. Every HTML document ought to have something like this boilerplate in it. It is not required to compose this boilerplate by hand because it may be automatically generated by the majority of development environments or more powerful text editors.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Document</title>
</head>
<body>
```

```
</body>

</html>
```

1.1.1.2 Tags and Elements

HTML tags and elements are essential components of HTML documents. A tag is a piece of HTML code that instructs a web browser how to display content on a webpage. In contrast, an element is made up of one or more tags and the content they enclose.

Tags are enclosed in angle brackets `< >` and usually come in pairs, with an opening tag and a closing tag. The opening tag begins with the name of the tag, followed by any attributes in the form of name-value pairs, and ends with a closing angle bracket `>`. The closing tag begins with a forward slash `/`, followed by the tag name, and ends with a closing angle bracket `>`.

For example, to create a paragraph element, we use the `<p>` tag to indicate the start of the element and `</p>` to indicate the end of the element. The text we want to include in the paragraph is enclosed between these two tags.

```
<p>This is a paragraph element</p>
```

HTML has a wide range of tags that can be used to structure content, including headings, paragraphs, lists, tables, forms, and more. Each tag has a specific purpose and should be used appropriately to create well-structured and semantically meaningful web pages.

It is important to note that not all HTML elements require a closing tag. For example, self-closing elements like `<img>` and `<br>` don't require a closing tag.

```
<img src="image.jpg" alt="An example image">
```

In this example, the `<img>` tag is a self-closing tag that displays an image on the web page. The `src` attribute specifies the URL of the image file, and the `alt` attribute provides a text description of the image.[7][8]

When using HTML tags and elements, it's important to keep in mind the proper use of tags to create clean and semantic code.

1.1.1.3 Head Element

The head element in HTML is a container that holds a web page's metadata. This metadata includes information that the browser requires to display the page correctly, such as the title of the page, links to stylesheets and scripts, and other information that can improve the accessibility and search engine optimization (SEO) of the page. Examples of this information include the title of the page, links to stylesheets and scripts, and other relevant information.

An overview of some of the most important tags that are utilized within the head element is as follows:

- Title Element:

  The title element is responsible for defining the title of the HTML page, which is shown in the top bar of the browser and is also utilized by search engines to determine the subject matter of the page. As demonstrated in the following example, the title element needs to be positioned inside the head element:

  ```
  <head>
   <title>My Page Title</title>
  </head>
  ```

- Meta Element:

  The meta element is a component that can be used to supply metadata about an HTML page. The "name" and "content" attributes of the meta element are the ones that are utilized the most frequently. The "name" attribute is used to determine the kind of metadata that is being provided, but the "content" attribute is responsible for supplying the actual metadata. The character set that the website is using, a description of the page written specifically for search engines, and keywords that describe the content of the page are all examples of the types of metadata that can be provided by utilizing the meta element. Here is an illustration of how to use the meta element:

```
<head>
 <meta charset="UTF-8">
<meta name="description" content="This is a description of my web page">
 <meta name="keywords" content="HTML, CSS, JavaScript">
 <meta name="author" content="Layth Al-Zamili">
</head>
```

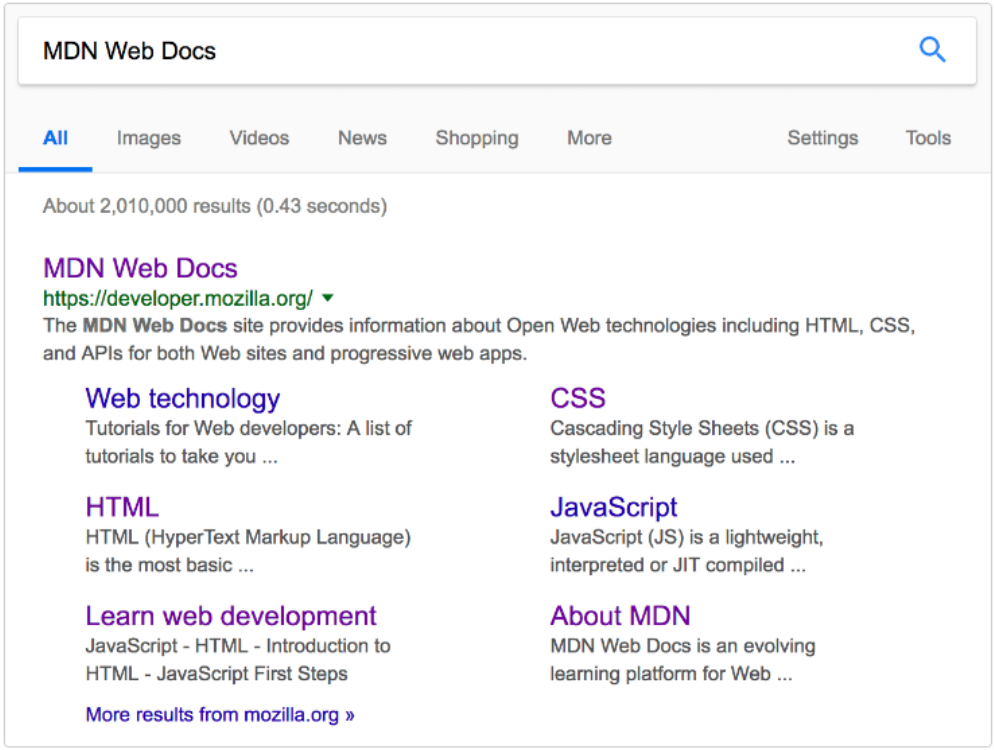

Figure 2 Shows the description <meta>and <title>[10].

- Link Element: The link element is used to link external resources to the HTML page. These external resources can include stylesheets, icons, and other files. The "href" attribute of the link element, which indicates the location of the external resource, is the one that is utilized the link element the most frequently. The following is an example of linking a stylesheet to an HTML page by using the link element:

```
<head>
<link rel="stylesheet" type="text/css" href="style.css">
</head>
```

- Script Element: The script element allows JavaScript code to be embedded into an HTML page. The "src" attribute of the script element, which indicates the location of the JavaScript file, is the one that is utilized the vast majority of the time. The following is an example of linking an HTML page to a JavaScript file by utilizing the script element:

```
<head>
<script src="script.js"></script>
</head>
```

- Base Element:

  The base URL for all relative URLs present within the page can be set with the help of this tag.[9]

```
<head>
 <base href="https://webpage.com/" />
</head>
```

  In overview, the head element in HTML supplies essential metadata that is required for the browser to correctly display the page and for search engines to recognize and index the content of the page. These two functions are dependent on each other. Web developers are able to optimize their pages for search engine optimization (SEO), user experience (UX), and accessibility by using features such as titles, meta descriptions, links, and scripts.

1.1.1.4 Body Element

An HTML document is not complete without the body element, which is responsible for defining the primary content of a web page. This element is an essential component. The body tag is one of the fundamental HTML tags, and it houses all of the material that can be seen on a web page. This content includes not only text but also photos, videos, and other forms of multimedia.

The head tag, which includes metadata about the web page like as the title, author, and description, must come before the body tag, which must be used after the head tag. The body tag is where all of the visible material is located, as opposed to the head tag, which does not contain any visible text.

The following is an illustration of one possible utilization of the body tag in HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>My Website</title>
</head>
<body>
  <h1>Welcome to My Website</h1>
  <p>This is my first paragraph. </p>
  <img src="image.jpg" alt="My Image">
  <p>This is my second paragraph. </p>
</body>
</html>
```

Following the h1 tag that specifies the primary heading of the web page that this example is demonstrating, there are two paragraphs of text and an image that are contained within the body tag. The img tag is used to define the picture, while the alt attribute supplies alternate text for viewers who are unable to view the image.

It is absolutely necessary to correctly format the content that is contained within the body tag. This not only makes the information easier to read and more accessible to consumers, but it also makes it easier for search engines to interpret the content, which can lead to an improvement in the website's ranking within search engines.

The following are some suggestions or tips to help format the content that is contained within the body tag:

- Use headings and subheadings to break up the content into sections and make it easier to read.

- Use paragraphs to break up long blocks of text and make it easier to skim the content.
- Use lists to present information in a structured format, such as bullet points or numbered lists.
- Use images and other multimedia elements to illustrate the content and make it more engaging.

In overview, the body tag is a crucial part of an HTML document, and it contains all the visible content of a web page. Structuring the content within the body tag correctly is essential to make the content more readable and accessible to users and improve the website's search engine rankings.

1.1.2 Cascading Style Sheets(CSS)

CSS is a style sheet language used to describe the look and formatting of an HTML document. It is used to separate the presentation of the document from its content, making it easier to maintain and update the styling of a website.

Here are some sections explaining the key concepts of CSS:

1.1.2.1 Selectors

Selectors are used to target specific HTML elements and apply styles to them. Selectors can target elements based on their tag name, class, ID, or other attributes. Here are some examples of CSS selectors:

```css
/* Target all h1 elements */
h1 {
  color: red;
}

/* Target all elements with class 'button' */
.button {
  background-color: blue;
}

/* Target an element with ID 'header' */
#header {
  font-size: 24px;
}
```

Figure 3 CSS Selectors

1.1.2.2 Properties and Values

CSS properties define the specific styles that are applied to the targeted elements. Each property has a value that determines how the style is applied. Here are some examples of CSS properties and their values:

```
/* Set the font size to 16 pixels */
font-size: 16px;

/* Set the background color to red */
background-color: red;

/* Set the border to a solid black line */
border: 1px solid black;
```

Figure 4 CSS properties with values

1.1.2.3 Cascading and Inheritance

CSS stands for "Cascading Style Sheets," and the "cascading" part refers to the way that styles are applied to HTML elements. Styles are applied in a specific order, and if there are conflicting styles, the most specific style wins.[11][12]

Inheritance is another important concept in CSS. If a style is applied to a parent element, it will be inherited by its child elements. For example, if you set the font family on the body element, all the text within the body element will inherit that font family by default.

Here is an example of a CSS file from my current project:

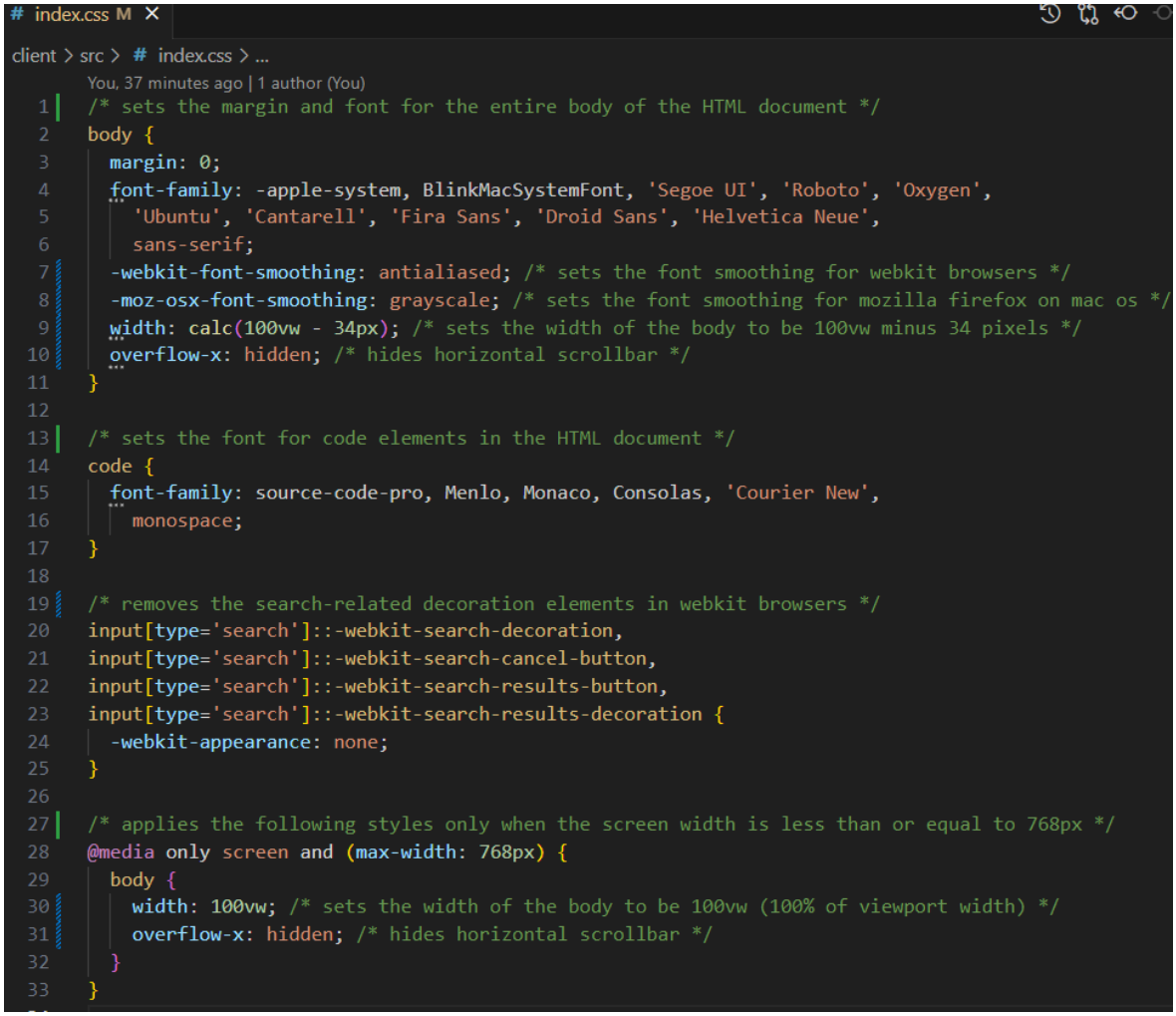

Figure 5 CSS code from project

In the above figure is a block of Cascading Style Sheets (CSS) code, which is used to define the presentation of an HTML document. The first part of the code sets the margin, font, font smoothing, width, and overflow properties for the entire body of the HTML document. It also defines a specific font for code elements in the document and removes certain decoration elements for search input in webkit browsers. The second part of the code uses a media query to apply specific styles to the body when the screen width is less than or equal to 768px. In this case, the width of the body is set to 100% of the viewport width and horizontal scrollbar is hidden.

In overview, CSS is a powerful tool for styling HTML documents and separating the presentation from the content. Understanding selectors, properties and values, cascading and inheritance, and how to write CSS code can help you create visually appealing and engaging websites.

1.1.3 JavaScript

JavaScript was created by Brendan Eich while he was working at Netscape Communications in 1995[13]. The goal of the language was to enable dynamic interactions on web pages, and it quickly became popular due to its ability to create interactive web applications. In 1997, JavaScript was standardized by the European Computer Manufacturers Association (ECMA), and the standardization led to the creation of ECMAScript, which is the official name for the language that is commonly referred to as JavaScript.[13]. The core JavaScript language and its built-in datatypes are the subject of international standards, and compatibility across implementations is very good. Parts of client-side JavaScript are formally standardized, other parts are de facto standards, and other parts are browser-specific extensions. Cross-browser compatibility is often an important concern for client-side JavaScript programmers [15].

1.1.3.1 Basics of JavaScript

JavaScript is a programming language that can add interactivity to your web pages and web applications in several ways [14]. It can be used to add dynamic behavior to web pages, validate form inputs, create animations, and much more. Here are some key concepts of JavaScript:

- Variables:

  Variables are used to store data in JavaScript. They are capable of storing a variety of data, including strings, numbers, and boolean values, among others. Consider the following example of a variable as an illustration:

  ```
  var name = "John";
  ```

- Functions:

  Code that carries out a certain task is grouped into functions. They are handy for reusing code because a script can call them several times, which makes them valuable in general. Consider the following illustration of a function declaration:

  ```
  function addNumbers(numb1, numb2) {
   return numb1 + numb2;
  }
  ```

- Objects:

  Objects are utilized to organize and group together similar data and functions. They are able to generate more complicated data structures, in addition to having the ability to have properties and methods. One example of an object is as follows:

```
var adult = {
  name: "Layth",
  age: 31,
  sayHello: function() {
    console.log("Hello!");
  }
};
```

- Control Flow:

  Control flow is a technique that allows programmers to dictate the order in which individual lines of code are run. JavaScript is capable of supporting conditional statements (if/else), loops (for/while), and switch statements. This is an illustration of a conditional statement:

```
if (age < 18) {
  console.log("Sorry, you are not old enough to enter.");
} else {
  console.log("You can enter");
}
```

- Example Code:

  An example of a function written in JavaScript that checks the validity of a form input is as follows:

```
function validateForm() {
  var email = document.getElementById("email").value;
  var password = document.getElementById("password").value;
  if (email == "") {
    alert("Email field is required.");
    return false;
  }
  if (password == "") {
    alert("Password field is required.");
    return false;
  }
  return true;
}
```

1.1.3.2 Advanced JavaScript Concepts:

JavaScript is a versatile and powerful programming language that offers a wide range of advanced concepts and features. Let's explore some of them:

- Prototypes and Object-Oriented Programming (OOP):

  JavaScript is a prototype-based language, which means objects can inherit properties and methods from other objects. This concept is fundamental to understanding

  JavaScript's approach to object-oriented programming (OOP). Here's an example:

```javascript
// Create a constructor function
function Person(name, age) {
  this.name = name;
  this.age = age;
}

// Add a method to the prototype
Person.prototype.sayHello = function() {
  console.log(`Hello, my name is ${this.name}`);
};

// Create an instance of the Person object
var john = new Person("John", 30);
john.sayHello(); // Output: Hello, my name is John
```

Figure 6 OOP project

In the example above shown in figure (6) the code defines a constructor function called `Person` that creates objects with `name` and `age` properties. It adds a method `sayHello` to the prototype of the `Person` object, which allows instances of `Person` to access and invoke the method. An instance of `Person` named `john` is created, and the `sayHello` method is called on `john`, printing a greeting message to the console.

- Asynchronous JavaScript and Promises:

  JavaScript is designed to handle asynchronous operations, such as sending requests over the network or reading and writing files. Asynchronous operations are non-blocking, which means that other code can continue to execute even while the asynchronous operation is taking place. The use of promises is a powerful technique to manage the results of asynchronous operations and to handle them. Take a look at this example:

```
function fetchData() {
  return new Promise(function(resolve, reject) {
    setTimeout(function() {
      var data = { message: "Data fetched successfully" };
      resolve(data);
    }, 2000);
  });
}

fetchData()
  .then(function(data) {
    console.log(data.message);
  })
  .catch(function(error) {
    console.log("Error:", error);
  });
```

Figure 7 Asynchronous operations

In the example above shown in figure (7) provided code uses Promises to fetch data asynchronously. The `fetchData` function returns a Promise that resolves after a 2-second delay. Inside the Promise, a timeout function is used to simulate an asynchronous operation. When the timeout expires, the Promise resolves with an object containing a `message` property set to "Data fetched successfully" using the `resolve` function.

After calling `fetchData()`, the `.then()` method is used to handle the resolved Promise. It takes a callback function that logs the `message` property of the resolved data to the console using `console.log(data.message)`.If an error occurs during the Promise execution, the `.catch()` method is used to handle it. It takes a callback function that logs an error message to the console using `console.log("Error:", error)`.In summary, this code fetches data asynchronously using Promises. Once the data is fetched successfully, the resolved data's `message` property is logged to the console. If an error occurs, it is caught, and an error message is logged.

- Modules and Module Bundlers:

  Modularity is an important concept in JavaScript, allowing you to split your code into reusable and maintainable modules. Modules encapsulate related code and provide a way to import and export functionality between files. Module bundlers like

webpack or Parcel help combine modules into a single file for deployment in a browser. Here's an example of exporting and importing modules:

```
// math.js
export function add(a, b) {
  return a + b;
}

export function subtract(a, b) {
  return a - b;
}
```

```
// app.js
import { add, subtract } from "./math.js";

console.log(add(5, 2)); // Output: 7
console.log(subtract(5, 2)); // Output: 3
```

Figure 8 exporting and importing modules.

In this example shown in figure (8), I have a module ‘math.js’ that exports two functions, ‘add’ and ‘subtract’. In ‘app.js’ ,I import these functions using the ‘import’ statement and use them in our code.

1.1.4 React.js

Since we'll be exploring React in depth later, let's provide a general overview for now, React.js, commonly referred to as React, is a popular framework written in JavaScript that is used to create front-end apps. It was initially developed by Facebook, and it has since gained widespread use because it enables programmers to rapidly create apps by employing a programming paradigm known as JSX [16], which connects JavaScript with a syntax that is similar to that of HTML When React was released in 2013 the web development community was both interested and seemingly disgusted by what React was doing [17]. React is capable of resolving a particular group of issues as well as one issue in general. According to Facebook and Instagram, React creates expansive user interfaces using dynamic data. Many web developers may be able to relate to large-scale user interfaces with data that varies over time from their own professional or recreational coding experiences. In the context of

contemporary web development, HTML, CSS, and JavaScript are frequently used to outsource most of the responsibility for the user interface. The common request/response to the server is constrained in these apps, which are also known as single-page applications, in order to highlight the capabilities of the browser. Given that the majority of browsers can handle complicated layout and interactivity, this is only natural. When the code from your weekend project becomes unmaintainable, a problem occurs. To make the data bind properly, you must "bolt on" extra pieces of code. When a secondary business requirement unintentionally breaks how the user interface displays a few interactions after the user begins a task, you may need to reorganize an application. All of this results in user interfaces that are delicate, intricate, and difficult to maintain. All of these issues are what React aims to fix. Consider the earlier mentioned client-side Model-View-Controller architecture with two-way data binding in templates. The views in this application must listen to the models and independently change their presentation in response to user interaction or model changes. This is not an obvious performance constraint in a simple program, which is more significant for developer productivity. As more models and views are included in the program, its size will certainly increase. A delicate and intricate web of code that connects the views to their models controls the relationships between each of them. This gradually gets trickier and trickier. Deeply nested objects or objects located in distant models are now having an impact on other objects' output. Because maintaining a tracking technique becomes increasingly challenging, an update that occurs may not even be fully known by the developer. This makes writing and testing your code more challenging, which makes it more challenging to create and publish a new method or feature. Development time has increased dramatically, and the code is now less predictable. React specifically seeks to address this issue. React began as a hypothetical exercise. Facebook reasoned that since the original layout code describing how the program should look had already been written, why not just run the startup code once the data or state changed in the application? You're probably cringing right now because you understand that doing this would compromise performance and the user experience would be compromised. You will notice screen flickers and flashes of unstyled content when you totally replace the code in a browser. It will merely seem ineffective. Facebook was aware of this but also highlighted that what it had developed—a system for updating the state when data changes—was somewhat effective. Facebook then came to the conclusion that it would have a solution if the replacement process could be streamlined. React was created in this way to address a particular set of issues[17].

1.1.4.1 Basics of React.js

Because React.js is predicated on the idea of reusable components, it enables the user interface to be partitioned into numerous discrete elements that can function independently of one another. The following is a list of important React.js concepts:

1.1.4.1.1 Components:

In React, a user interface is constructed from components. A component may consist of a minor portion of a page (such as a button) or the entire page. Components encapsulate their own logic and may be utilized across an application. Using either JavaScript classes or functions, React components can be created. React components can be divided into two main types: functional components and class components [18].

I. Functional Components:

Functional components are JavaScript functions that take props (properties) as parameters and return JSX (JavaScript XML) elements that define the output of the component. They are the simplest type of React component and are extensively used due to their simplicity and testing convenience.

a. When working with functional components, keep the following best practices in mind:

- Maintain component focus: Functional components should concentrate on rendering a specific UI element or functionality and have a single responsibility.
- Use destructuring for props: Destructuring props in functional components makes the code cleaner and easier to read.
- Avoid modifying props directly: Treat props as read-only and avoid directly modifying them within a functional component.
- Use memoization: Use the React.memo higher-order component or the useMemo hook to memoize expensive computations and avoid unnecessary re-rendering.
- Use hooks for state and side effects: Utilize hooks like useState, useEffect, and others to manage state and perform side effects within functional components.

b. Here are the benefits of Functional Components:

- Simplicity: Functional components have a simpler syntax compared to class components. They are plain JavaScript functions that take props as input and return JSX elements as output.
- Easy to understand: Functional components are easier to understand and reason about because they don't have their own internal state or lifecycle methods.

- Reusability: Functional components are highly reusable. They can be easily composed and used in other components.
- Performance: Functional components have a smaller memory footprint and are generally faster to render than class components. They avoid the overhead of managing a component instance and lifecycle. When writing new code, we advise switching from utilizing class components to using function components.[23]

In overview, functional components are a more straightforward and lightweight approach to build components in React.js. They are JavaScript functions that take parameters called props and return JSX elements that specify the output of the component. The benefits of functional components include enhanced performance, reusability, simplicity, and ease of understanding. React hooks gave functional components the ability to handle state, lifecycle methods, and other React features that were previously only available in class components. Functional components can manage state and carry outside effects with the use of hooks like useState, useEffect, and others. Keep functional components focused while working with them, use destructuring for props instead of directly altering them, use memoization for speed improvement, and use hooks for side effects and state management.

Overall, because of their simplicity, reusability, and performance advantages, functional components are the suggested method for creating components in React.js. They offer a simple and direct method for developing UI elements in React apps.

Below will be a live example of functional component from the project :

```
// Importing React and the Error404 SVG component as a React component
import React from 'react';
import { ReactComponent as Error404 } from '../svg/404-error.svg';
// Importing Typography, SvgIcon, and ErrorOutlineIcon components from Mate-
rial-UI
import { Typography, SvgIcon } from '@material-ui/core';
import ErrorOutlineIcon from '@material-ui/icons/ErrorOutline';
// Defining the ErrorPage component with a parameter 'errorMsg'
const ErrorPage = ({ errorMsg }) => {
  // Checking if the error message includes specific strings to determine if
it's a "Not Found" error
  const isNotFoundError =
    errorMsg.includes('does not exist') || errorMsg.includes('Malformat-
ted');
  // Returning the JSX markup
  return (
    <div style={{ textAlign: 'center', marginTop: '20%' }}>
      {isNotFoundError ? (
        // Displaying the Error404 SVG icon if it's a "Not Found" error
        <SvgIcon
          color="primary"
          style={{ fontSize: '8em', marginBottom: '0.5em' }}
        >
          <Error404 />
        </SvgIcon>
      ) : (
        // Displaying the ErrorOutlineIcon if it's not a "Not Found" error
        <ErrorOutlineIcon
          color="primary"
          style={{ fontSize: '8em', marginBottom: '0.5em' }}
        />
      )}
      <Typography color="secondary" variant="h4">
        {/* Displaying "404 Not Found" if it's a "Not Found" error, other-
wise displaying "Error" */}
        {isNotFoundError ? `404 Not Found` : 'Error'}
      </Typography>
      <Typography color="secondary" variant="h6">
        {/* Displaying the error message */}
        {errorMsg}
      </Typography>
    </div>
  );
};
// Exporting the ErrorPage component as the default export
export default ErrorPage;
```

Figure 9 functional component from the project.

This code above in Figure (9) defines a React component called `ErrorPage` that displays an error message along with an icon. Here's a breakdown of what the code does:

1. Importing dependencies:

- The code imports `React` from the 'react' package. This is necessary to define and use React components.
- The code also imports the `Error404` component as a React component from the '../svg/404-error.svg' file. This component is an SVG icon representing a 404 error.
- Additionally, the code imports `Typography`, `SvgIcon`, and `ErrorOutlineIcon` components from the Material-UI library. These components are used to style and display text and icons.

2. Defining the `ErrorPage` component:

- The `ErrorPage` component is defined as a functional component that takes an `errorMsg` parameter.
- Inside the component, it checks if the `errorMsg` includes specific strings to determine if it's a "Not Found" error. It sets the `isNotFoundError` variable to `true` if the error message contains either "does not exist" or "Malformatted".

3. Returning the JSX markup:

- The component returns JSX markup, which represents the structure and content of the component.
- The JSX is wrapped inside a `<div>` element with inline styles to center the content vertically and add some top margin.
- Depending on the value of `isNotFoundError`, the component conditionally renders either the `Error404` SVG icon or the `ErrorOutlineIcon`.
- Both icons are wrapped in either a `SvgIcon` or an `ErrorOutlineIcon` component, respectively. These components provide styling and configuration options for the icons.
- Below the icon, the component displays a `Typography` component with the color set to "secondary" and the variant set to "h4". The text of this component is

conditionally set to either "404 Not Found" or "Error" based on the `isNotFoundError` variable.

- Below the title, another `Typography` component is used to display the `errorMsg` passed as a prop.

4. Exporting the `ErrorPage` component:

- The `ErrorPage` component is exported as the default export of the module, which means it can be imported and used in other parts of the application.

In summary, the `ErrorPage` component is a reusable component that displays an error message along with an appropriate icon based on the type of error. It uses Material-UI components for styling and React components for composition.

II. Class Components:

Class components are JavaScript classes that inherit "React. Component" as their root class. Prior to the introduction of hooks in React 16.8, they were the primary method to create components and have more features than functional components [19]. Class components have their own internal state and lifecycle methods, making them appropriate for logic and interactions that are more complex. You can hook into various stages of a class component's lifecycle by utilizing its lifecycle methods, which are provided by class components. These methods can be overridden in your class component to carry out tasks at particular moments, such as initializing the state, fetching data, or cleaning up resources. Here are some commonly used lifecycle methods in class components:

- 'componentDidMount': This method is called after the component has been rendered into the DOM. It is commonly used for fetching data from an API, setting up subscriptions, or initializing timers. If you implement componentDidMount, you will typically need to implement additional lifecycle methods in order to ensure that your application is free of errors. React will execute the ‘componentDidMount’ method when your component is added (mounted) to the screen if you define it [20].

Below an example about 'componentDidMount' method

```
componentDidMount() {
// Perform actions after the component is mounted
// For example, fetch data from an API
  fetchData().then((data) => {
    this.setState({ data });
  });
}
```

- 'componentDidUpdate(prevProps, prevState)': This method is called after the component has been updated. It allows you to perform actions when the component's props or state have changed. This method is not invoked for the first render [21]

  Here is an example of 'componentDidUpdate' method :

```
componentDidUpdate(prevProps, prevState) {
  // Perform actions when props or state have changed
  // For example, update the component based on new props
  if (this.props.value !== prevProps.value) {
    this.setState({ value: this.props.value });
  }
}
```

- 'componentWillUnmount': Before the component is unmounted and removed from the DOM, this method is invoked. It is utilized to clear up resources including event listeners, timers, and subscriptions. The logic contained within componentWillUnmount must "mirror" the logic contained within componentDidMount [22]

  Here is an example of 'componentWillUnmount method :

```
componentWillUnmount() {
  // Clean up resources before the component is unmounted
  // For example, unsubscribe from a subscription
  this.subscription.unsubscribe();
}
```

```
import React from 'react';  7.6k (gzipped: 3k)

You, 1 second ago | 1 author (You) | Complexity is 6 It's time to do something...
class Counter extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0,
    };
  }

  increment() {
    this.setState(prevState => ({
      count: prevState.count + 1,
    }));
  }

  Complexity is 6 It's time to do something...
  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={() => this.increment()}>Increment</button>
      </div>
    );
  }
}

export default Counter;
```

Figure 10 Class component.

1.1.4.1.2 JSX (JavaScript XML) :

JSX is a syntax extension for JavaScript that makes it possible to write code in JavaScript that looks and feels like HTML. It is one of the most important aspects of React.js and offers a method that is both succinct and declarative for defining the structure as well as the appearance of UI components. You can write tags that are similar to HTML directly in your JavaScript code if you are using JSX. These tags will then be translated into react elements. Due to the fact that the syntax is quite similar to HTML, this makes it much simpler to perceive and comprehend the structure of the UI components you have. The majority of React developers like JSX due to its compact nature, and the majority of codebases make use of it [24].Here is a live example of JSX from the project :

```jsx
import React from 'react';

import { Alert, AlertTitle } from '@material-ui/lab';
import { useAlertStyles } from '../styles/muiStyles';

const AlertMessage = ({ severity, error, clearError }) => {
  const classes = useAlertStyles();

  if (!error) {
    return null;
  }

  return (
    <div className={classes.root}>
      <Alert severity={severity} onClose={clearError}>
        <AlertTitle>Error</AlertTitle>
        {error}
      </Alert>
    </div>
  );
};

export default AlertMessage;
```

Figure 11 Example of JSX from the project

In the example above in figure (11) that the code I provided is written in JSX and that have a functional component named `AlertMessage` that receives props `severity`, `error`, and `clearError`. It imports `React` and some components from the `@material-ui/lab` library. It also imports a custom hook `useAlertStyles` from a file `../styles/muiStyles`. The `AlertMessage` component conditionally renders an `Alert` component from `@material-ui/lab` based on the `error` prop. If `error` is falsy, the component returns `null` and doesn't render anything. Otherwise, it renders the `Alert` component with the specified `severity` and `onClose` handler. The error message is rendered within the `Alert` component using the `AlertTitle` and the `error` prop. The `classes` variable is assigned the value returned by the `useAlertStyles` hook, which is likely a CSS class object generated by `@material-ui/styles` to apply styles to the `AlertMessage` component. Overall, the code demonstrates the usage of JSX syntax to define a React component that displays an alert message with an error using Material-UI's `Alert` component.

I. Benefits of JSX :

- Declarative Syntax: JSX provides you with the ability to specify the structure as well as the appearance of your user interface in a declarative manner. It is intuitive and easier to read and write compared to manually creating elements using 'React.createElement'.
- JavaScript Integration: Since JSX is a syntax extension of JavaScript, it seamlessly integrates with JavaScript code. You can embed JavaScript expressions and logic within JSX using curly braces `{}`. This enables dynamic rendering and data manipulation.
- Component Reusability: JSX facilitates the creation of reusable components. You can create custom components and compose them together to build complex UIs, just like composing HTML tags.
- Static Type Checking: JSX supports static type checking through tools like TypeScript and Flow. This helps catch type-related errors during development and improves code robustness.

II. JSX Transformation :

JSX, as a syntax extension for JavaScript, needs to be transformed into regular JavaScript code before it can be executed by the browser or JavaScript runtime. This transformation is handled by a tool called a transpiler, such as Babel. The JSX transformation process involves converting JSX elements into `React.createElement` function calls. This function creates React elements that represent the components. The transformation occurs during the build process of a React application. Here's an example to illustrate the JSX transformation process:

```
// JSX code
const element = <h1>Hello, World!</h1>;
```

After the transformation, the JSX code is converted to the following JavaScript code:

```
// Transformed JavaScript code
const element = React.createElement('h1', null, 'Hello, World!');
```

In the transformed code, the JSX element `<h1>Hello, World!</h1>` is converted into a `React.createElement` function call. The function takes three arguments:

- The type of the element, in this case, the string `'h1'` representing the HTML heading tag.
- An object of properties or attributes to be applied to the element. Since there are no attributes in the example, `null` is passed as the second argument.
- The children of the element, which in this case is the string `'Hello, World!'`.

The `React.createElement` function then creates a React element with the specified type, properties, and children. This element can be further processed and rendered by React to generate the corresponding HTML representation on the web page. It's important to note that the JSX transformation is performed by a transpiler, such as Babel, which needs to be configured in the development environment to handle JSX syntax. The transformed code is what gets executed by the browser or JavaScript runtime. Understanding the JSX transformation process helps in developing React applications and enables the use of JSX syntax to create expressive and readable UI components.

1.1.4.1.3 Virtual DOM:

The rendering performance of web applications can be improved by using React.js Virtual DOM, which stands for "Document Object Model," which is both a concept and a technology. It is a lightweight representation of the UI components and their structure, and it is an abstraction of the actual DOM that is used by the browser.

I. How the Virtual DOM Works :

- Initial Render: When a React component is first rendered, it generates a virtual representation of the component's UI structure, known as the Virtual DOM. This virtual representation is a JavaScript object tree that mirrors the actual DOM structure.
- Diffing Algorithm: Whenever there are changes to the component's state or props, React re-renders the component and generates a new Virtual DOM representation. It then compares the new Virtual DOM with the previous one using a process called "diffing.". The diffing algorithm efficiently identifies the differences between the new and previous Virtual DOM trees. By analyzing the changes, React determines the minimal number of updates required to synchronize the actual DOM with the new Virtual DOM.
- 3. Efficient Updates: Once the differences are identified, React updates only the necessary parts of the actual DOM to reflect the changes. This process is often referred to as "reconciliation.". React's reconciliation algorithm optimizes the update process by minimizing the number of actual DOM manipulations required. Instead of

updating each individual element, React calculates the most efficient way to update the DOM based on the identified differences in the Virtual DOM.

II. Benefits of the Virtual DOM

The Virtual DOM provides several benefits:

- Performance Optimization: By using the Virtual DOM and its efficient diffing algorithm, React minimizes the number of actual DOM updates. This leads to improved rendering performance, especially when dealing with complex UI structures or frequent updates.
- Developer Productivity: The Virtual DOM allows developers to work with a simplified and declarative programming model. They can focus on updating the component's state and let React handle the efficient update process.
- Cross-Platform Compatibility: Since the Virtual DOM is an abstraction layer, it provides a consistent interface across different platforms and browsers. This allows React applications to work seamlessly across various environments.

In overview, React.js uses the Virtual DOM approach to enhance the speed at which web applications render. It serves as a simple representation of the UI components and their structure and is an abstraction of the actual browser DOM.React reduces the number of actual DOM updates needed when a component's state or props change by utilizing the Virtual DOM. This is accomplished via a technique known as "diffing," which effectively recognizes the variations between the new and earlier Virtual DOM representations. Improved rendering efficiency, greater developer productivity, and cross-platform compatibility are all advantages of using the Virtual DOM. Based on the detected discrepancies in the Virtual DOM, React's reconciliation algorithm automatically chooses the most effective approach to update the actual DOM, producing optimized and streamlined changes. Developers can concentrate on changing the state of the component by understanding the Virtual DOM and relying on React to handle the quick update procedure. React guarantees quicker rendering and a more seamless user experience by minimizing needless DOM modifications [25]. A Note on DOM Events ,to avoid any confusion, a few clarifications are in order regarding the following line: onChange=(event => this.onTextChange(event))

React uses its own synthetic events system for performance (as well as convenienceand sanity). To help understand why, you need to consider how things are done in thepure DOM world[38] .

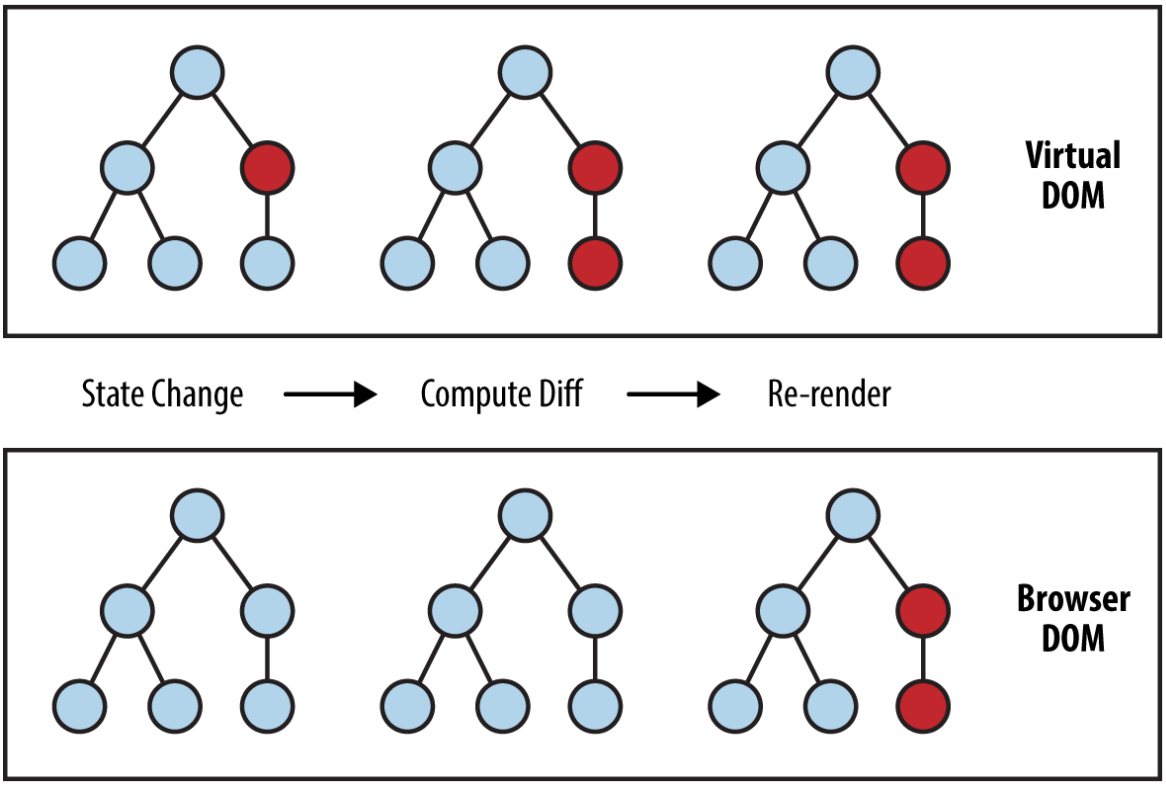

Figure 12 The virtual DOM tree and the diffing process [26].

1.1.4.1.4 State and Props :

State and props are two fundamental concepts in React.js that enable components to manage and communicate data. Properties are a mechanism for the outside world (users of the component) to configure your component,State is your internal data maintenance [38].

I. Props:

Props (short for properties) are inputs that are passed into a React component. They are similar to function arguments and provide a way to pass data from a parent component to its child components. Props are read-only and should not be modified within the child component [27].

Here's an example of how props are used:

```
// Parent component

import React from 'react';
```

```
import ChildComponent from './ChildComponent';
function ParentComponent() {
  const name = 'Layth';

  return <ChildComponent name={name} />;
}
export default ParentComponent;
// Child component
import React from 'react';
function ChildComponent(props) {
  return <h1>Hello, {props.name}!</h1>;
}
export default ChildComponent;
```

In this example, the `name` variable in the parent component is passed as a prop called `name` to the child component. The child component then receives the prop and uses it to render a greeting.

Props allow components to be reusable and provide a way to customize their behavior and appearance based on the data passed in from their parent components.

II. State :

State represents the internal data of a component. It is managed and controlled by the component itself. Unlike props, which are passed from parent components, state is self-contained within the component. To use state in a React component, you need to declare it using the `useState` hook or, in class components, by setting the initial state in the constructor. Updating state triggers a re-rendering of the component and its child components [27].

Here's an example of using state with the `useState` hook:

```
import React, { useState } from 'react';
function Counter() {
  const [count, setCount] = useState(0);
```

```
  const increment = () => {
    setCount(count + 1);
  };
 return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}
export default Counter;
```

In this example, the `useState` hook is used to define a state variable called `count` with an initial value of 0. The `setCount` function is used to update the value of `count` when the button is clicked.

State allows components to manage and track data that can change over time, such as user input, toggles, or API responses. It provides a way to handle dynamic behavior and update the UI accordingly.

III. Managing State and Props :

To manage state and props effectively, consider the following best practices:

- Props: Treat props as read-only within a component. They should be used for passing data from parent components and should not be modified within the child component.
- State: Update state using the appropriate methods provided by React, such as `useState` hook or `this.setState` in class components. Avoid directly modifying state using assignment (`=`).
- Data Flow: Data flows in a unidirectional manner in React, from parent components to child components via props. To modify the data, you can pass callback functions from parent components to child components, allowing child components to update the parent's state.

In overview, the essential ideas of state and props in React.js allow components to maintain and transmit data. Similar to function arguments, props allow data to be sent from parent components to child components. They are read-only and give you a means to alter how your

child components behave and look based on information you get from your parent components. Props enable the building of intricate UI structures and make components reusable. A component's state, which is handled and controlled by the component itself, reflects the internal data of the component. It enables components to manage dynamic behavior and adjust the user interface (UI). The component's internal state is self-contained, and it may be modified using the suitable React methods, such as the 'useState' hook or 'this.setState' in class components. React forces a re-rendering of the component and any child components when the state changes. Developers may create dynamic and interactive React applications by effectively managing and leveraging state and props. While state enables components to manage their own internal data and react to changes, props facilitate the transfer of data between components.

1.1.4.2 Advanced Concepts in React.js:

React.js provides a number of cutting-edge ideas and features that improve its capacity for creating intricate user interfaces. Let's examine a few of these ideas that used in my project:

1.1.4.2.1 React Hooks :

In React 16.8, React Hooks were added as a means to leverage state and other React features without having to write class components. Hooks give you a more terse and practical way to manage the state and side effects of a component while allowing you to reuse stateful functionality between components [28]. Here's a live example of a functional component using the useState and useEffect hooks from the project :

```
import React, { useState, useEffect, createRef } from 'react';
import { CircularProgress, Grid, Typography, InputLabel, MenuItem, FormCon-
trol, Select } from '@material-ui/core';

import PlaceDetails from '../PlaceDetails/PlaceDetails';
import useStyles from './styles.js';

const List = ({ places, type, setType, rating, setRating, childClicked,
isLoading }) => {
  const [elRefs, setElRefs] = useState([]);
  const classes = useStyles();

  useEffect(() => {
    // Create and initialize an array of refs with the length of 'places'
array
    setElRefs((refs) => Array(places.length).fill().map((_, i) => refs[i] ||
createRef()));
  }, [places]);

  return (
    <div className={classes.container}>
      <Typography variant="h4">Food & Dining around you</Typography>

      {isLoading ? (
        // Show loading spinner if isLoading is true
        <div className={classes.loading}>
          <CircularProgress size="5rem" />
        </div>
      ) : (
        <>
          {/* Render dropdown for selecting type */}
          <FormControl className={classes.formControl}>
            <InputLabel id="type">Type</InputLabel>
            <Select id="type" value={type} onChange={(e) => setType(e.tar-
get.value)}>
              <MenuItem value="restaurants">Restaurants</MenuItem>
              <MenuItem value="hotels">Hotels</MenuItem>
              <MenuItem value="attractions">Attractions</MenuItem>
            </Select>
          </FormControl>

          {/* Render dropdown for selecting rating */}
          <FormControl className={classes.formControl}>
            <InputLabel id="rating">Rating</InputLabel>
            <Select id="rating" value={rating} onChange={(e) => setRat-
ing(e.target.value)}>
              <MenuItem value="">All</MenuItem>
              <MenuItem value="3">Above 3.0</MenuItem>
              <MenuItem value="4">Above 4.0</MenuItem>
              <MenuItem value="4.5">Above 4.5</MenuItem>
            </Select>
          </FormControl>

          {/* Render grid of place details */}
          <Grid container spacing={3} className={classes.list}>
            {places?.map((place, i) => (
              <Grid ref={elRefs[i]} key={i} item xs={12}>
                <PlaceDetails selected={Number(childClicked) === i}
refProp={elRefs[i]} place={place} />
              </Grid>
            ))}
          </Grid>
        </>
      )}
    </div>
  );
};

export default List;
```

Figure 13 The useState and useEffect hooks.

The provided code above in figure (13) demonstrates the usage of the `useState` and `useEffect` hooks in React. Here's a summary of their functionality:

- **`useState`:**
  It is a hook that allows functional components to have local state. In the code, `const [elRefs, setElRefs] = useState([]);` initializes the `elRefs` state variable with an empty array and provides a function `setElRefs` to update its value.`elRefs` is used to store an array of references that will be used to target DOM elements.

- **`useEffect`:**
  It is a hook that enables performing side effects in functional components. In the code, `useEffect` is used to create and initialize an array of refs when the `places` dependency changes.The effect runs only when the `places` array is updated.The effect function updates the `elRefs` state by generating an array of refs using the `Array(places.length).fill().map()` method.This ensures that the `elRefs` state always corresponds to the number of `places`.

The rest of the code focuses on rendering the UI components, such as `CircularProgress`, `Typography`, `InputLabel`, `Select`, and `Grid`. These components are provided by Material-UI.

Overall, the code maintains a list of places and renders them along with dropdown menus for selecting the type and rating. It also handles the loading state and displays a loading spinner when necessary. The `PlaceDetails` component is rendered for each place in a grid layout.

1.1.4.2.2 React Router :

A popular library for handling routing in React apps is known as React Router. It makes it possible for you to develop a single-page application that has a number of different views or pages. You are able to handle dynamic URLs, browse between various views, and construct routes with the help of the React Router. The Create React App starter kit does not come with page routing. It is generally agreed that React Router is the best solution [29].

Installing React Router is the first step in using it, and you can do so by entering the following command into the directory containing your project which is I have already installed in my project:

```
npm install react-router-dom
```

Once installed, you can import the necessary components from the react-router-dom package [29] ,as shown below in the live example from my project :

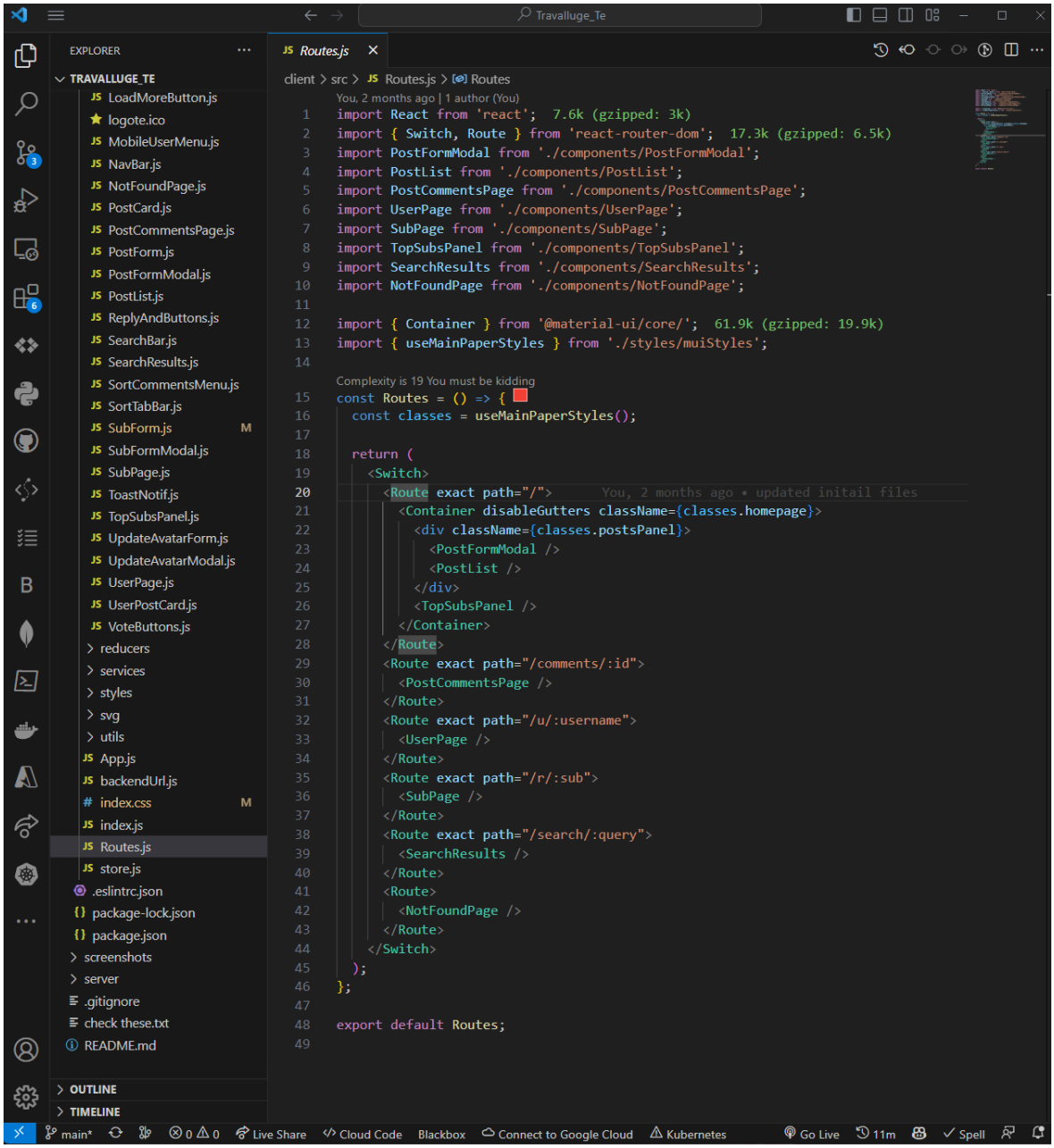

Figure 14 The React Router from the project.

The code I have provided in figure (14) defines the routing configuration for a React application using the React Router library. Here's an explanation of the code:

- Import Statements: The code imports necessary components from `react-router-dom` for routing, including `Switch` and `Route`. It also imports various components used for different routes, such as `PostFormModal`, `PostList`, `PostCommentsPage`, `UserPage`, `SubPage`, `TopSubsPanel`, `SearchResults`, and `NotFoundPage`. Additionally, it imports the `Container` component from `@material-ui/core` and a custom hook `useMainPaperStyles` from a custom styles file.
- Functional Component: The `Routes` component is a functional component that defines the routes and their corresponding components. Inside the component, the

`useMainPaperStyles` hook is used to get the CSS styles for the main container. The component returns a `Switch` component that renders only the first `Route` or `Redirect` that matches the current URL.

- Route Configuration: The `Switch` component wraps multiple `Route` components that define the different routes of the application. Each `Route` component has a `path` prop that specifies the URL path for which the route should be active. The `exact` keyword is used to ensure an exact match of the path. The `Route` components are associated with their corresponding components to be rendered when the URL matches the specified path. If a route does not match any of the defined paths, the last `Route` component without a `path` prop acts as a fallback and renders the `NotFoundPage` component.

Overall, the code defines the routes for different pages of the application using React Router. It specifies the components to render for each route and handles fallback scenarios when a route is not found. The imported components are used to build the UI for each route, and Material-UI's `Container` component is used for layout purposes.

1.1.4.2.3 State Management Libraries : Redux :

Redux has emerged as one of the clear winners in the field of Flux or Flux-like libraries. Redux is based on Flux, and it was designed to tackle the challenge of understanding how data changes flow through your application. Redux was developed by Dan Abramov and Andrew Clark. Since creating Redux, both have been hired by Facebook to work on the React team[39].

Redux is a JavaScript library for managing the state of applications. It is notably popular with the React framework. It contributes to the management of the application's state in a centralized and predictable manner. Redux makes use of a single immutable state tree and follows a style of data flow that is unidirectional. To use Redux with React, you need to install the `react-redux` package. It is important that you check that the React and Redux DevTools extensions have been successfully installed in your browser [30]. You can do this by running the following command in your project directory as I have done it in my project as well:

```
npm install react-redux
```

Once installed, you can import the necessary functions and hooks from `react-redux` , as shown below in the live example from my project :

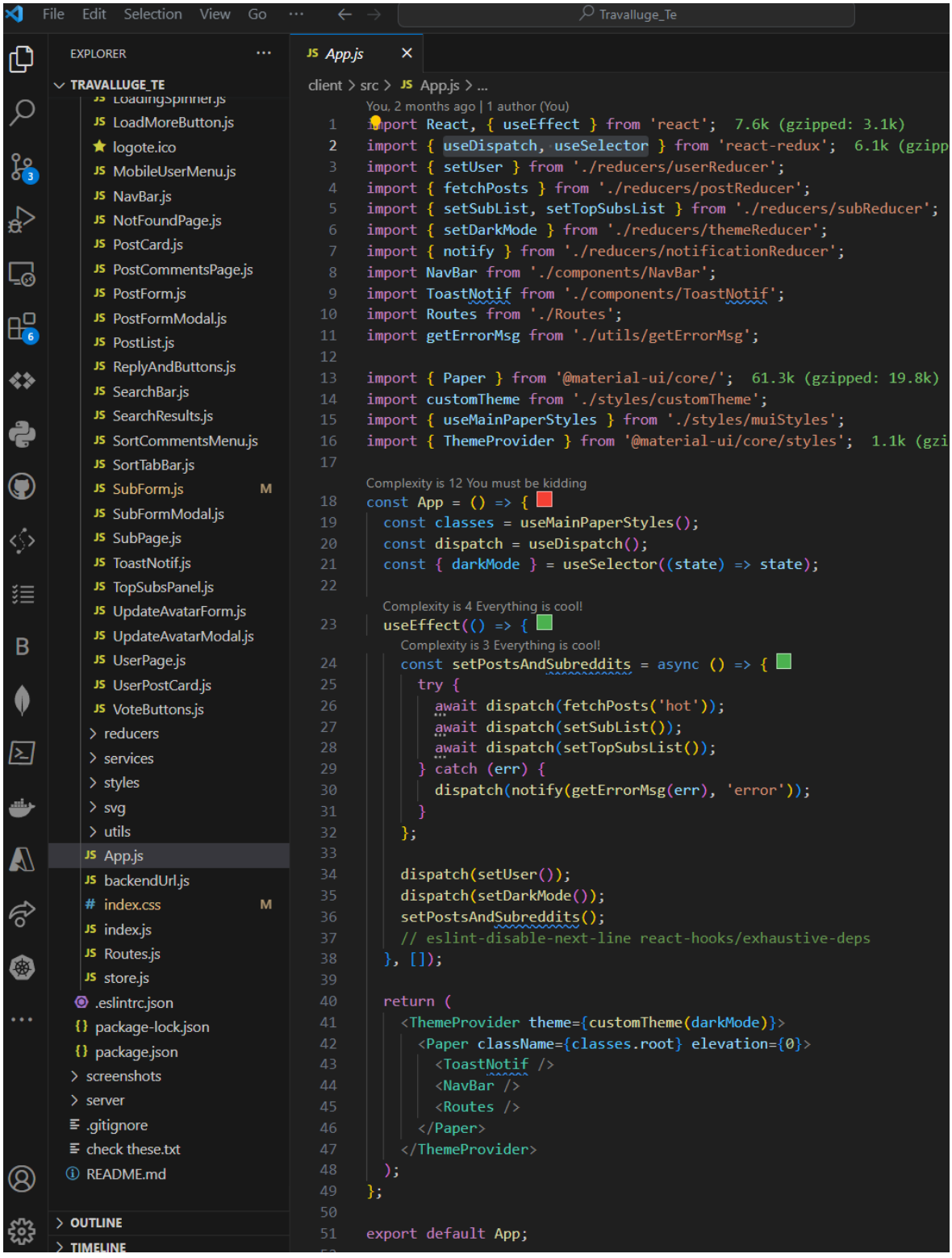

Figure 15 Redux with useDispatch & useSelector hooks from the project

The code above I have provided from my project represents the main component of a React application that uses Redux for state management. It imports necessary dependencies, such as React, Redux hooks (useDispatch and useSelector), and various reducer functions.

The code begins by importing the necessary reducer functions from separate files, such as userReducer, postReducer, subReducer, themeReducer, and notificationReducer. These reducers are responsible for managing different parts of the application's state.

Next, the code imports additional components and styling-related dependencies from Material-UI.

The `App` component is defined as a functional component. It begins by initializing variables using the `useMainPaperStyles` hook and the `useDispatch` and `useSelector` hooks from Redux. The `useDispatch` hook is used to obtain the dispatch function, which allows us to dispatch actions to update the state. The `useSelector` hook is used to retrieve the `darkMode` value from the Redux store.

Inside the `useEffect` hook, an asynchronous function called `setPostsAndSubreddits` is defined. This function is responsible for dispatching actions to fetch posts, set the subreddit list, and set the top subreddits list. If an error occurs during this process, it dispatches a notification with an error message.

The `setUser` and `setDarkMode` actions are also dispatched to initialize the user and dark mode settings.

Finally, the `setPostsAndSubreddits` function is called, and the `App` component returns the JSX code representing the application's UI. It uses the `ThemeProvider` component from Material-UI to apply a custom theme based on the `darkMode` value. The UI includes a `Paper` component with the defined CSS class (`classes.root`), and it renders the

`ToastNotif`, `NavBar`, and `Routes` components.

In summary, this code sets up the main component of a React application using Redux for state management. It dispatches various actions to initialize the state, fetch data, and render the UI components.

1.1.4.2.4 Form Handling: Formik :

Formik is a popular library in React.js that provides a simple and efficient way to handle forms. It simplifies the process of managing form state, validating user input, handling form submission, and displaying error messages . Also offers a declarative approach to form

handling in React applications. It integrates seamlessly with React components and provides a range of features to streamline the form management process. we will discuss the main features of Formik and how to use them :

- Installation and Importing:
  Installing Formik as a dependent in your project is necessary if you wish to make use of it. You are able to accomplish this by executing the following command:
  ```
  npm install formik
  ```
  Once installed, you can import Formik and other related components and hooks.
  ```
  import { Formik, Form } from 'formik';
  ```

- Form Structure:
  Within the `Formik` component, you define the structure of your form using the `<Form>` component. Inside the `<Form>` component, you can use various input components provided by Formik or create your own custom components. For example, the `<Field>` component represents an input field, and the `<ErrorMessage>` component displays validation errors for a specific field.
- Form Validation:
  Formik makes form validation easy by providing built-in validation support. You can define validation rules either by using the `validate` prop or by providing a `validationSchema` prop that utilizes the Yup validation library. In the `validate` function or Yup schema, you can define rules such as required fields, minimum or maximum length, regular expressions, and more.
- Form Submission:
  When the form is submitted, the `onSubmit` function provided to the `Formik` component is called. Inside this function, you can perform any necessary actions, such as making API calls, updating the application state, or navigating to a different page. Formik automatically handles form submission and prevents the default form behavior.
- Validation Errors:
  Formik automatically handles validation errors and displays them using the `<ErrorMessage>` component. It shows the error message associated with a specific field when the field is touched and its value is invalid.
- Form State and Values:
  Formik manages the state of the form, including values, errors, and submission status. You can access and manipulate the form state using various Formik hooks, such as `useFormikContext`, `useField`, and `useFormik`.
- Handling Input Fields:
  Formik provides the `<Field>` component to handle input fields. It can be used with various input types, such as text, email, password, checkboxes, radio buttons, and more. Additionally, you can define custom input components and use them with the `<Field>` component [31].

For ‘Formik’ component serves as the main wrapper for your form. It takes a set of props that allow you to configure its behavior and handle form-related operations. as shown below in the live example from my project for Formik :

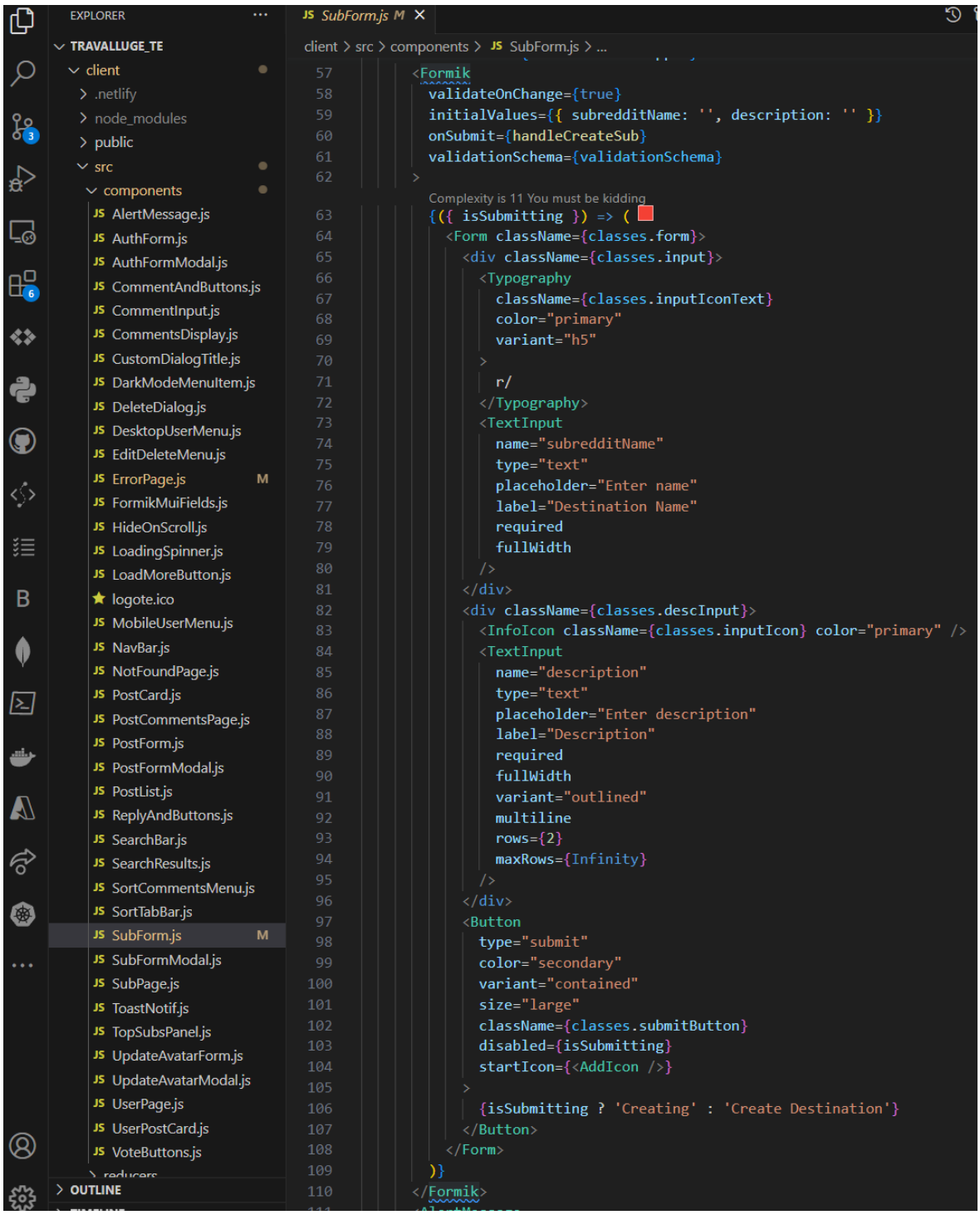

Figure 16 ‘Formik’ from the project

In overview, the provided code in figure (16) demonstrates the usage of the Formik library to create a form in a React application. Formik simplifies form handling, validation, and submission by providing a set of hooks and components. The code starts with the `<Formik>` component, which wraps the form and handles its state and validation. It specifies options like `validateOnChange`, `initialValues`, `onSubmit`, and `validationSchema` to control the form's behavior. Inside the function component within `<Formik>`, the actual form structure is defined using `<Form>`, `<Field>`, and other components. These components represent input fields, display validation errors, and define the form's layout. The form includes two input fields: one for the subreddit name and another for the description. The fields are rendered using `<TextInput>` components with various props like `name`, `type`, `placeholder`, `label`, `required`, and `fullWidth`.

Finally, the form includes a `<Button>` component that triggers the form submission. The button is conditionally rendered based on the `isSubmitting` prop, and its text is changed to reflect the current state of the form submission. By utilizing Formik, developers can easily handle form state, validation, and submission in their React applications, reducing boilerplate code and providing a convenient way to manage form-related operations.

1.1.4.2.5 Data Fetching: Axios :

Axios is a well-known JavaScript package used in React applications (and other JavaScript environments) for making HTTP calls, including retrieving data from APIs. It offers a straightforward and effective API for carrying out various request types, managing request and response interceptors, setting headers, and dealing with failures [32].

If you want to use Axios to make requests in your application then first you need to install it by using npm or yarn as shown below [32]:

```
npm install axios
```

or

```
yarn add axios
```

Here's an example of how you can use Axios library for making HTTP requests from my project :

Figure 17 Axios library for making HTTP requests from project

1.1.5 Material-UI

A well-liked user interface (UI) library for creating web apps is called Material-UI. It offers a selection of pre-designed, re-usable elements, styles, and themes that adhere to Google's Material Design standards. React is a JavaScript library for creating user interfaces, and Material-UI, which is popular among React users, is built on top of it.

Its huge library of pre-made UI components is one of Material-UI's main features. These elements include cards, modals, buttons, forms, navigation menus, and many more. These components allow developers to construct and integrate standard UI components into their applications more quickly and easily.

The stylistic possibilities provided by Material-UI go beyond the pre-designed components. The CSS-in-JS (JavaScript styling) method is used, and each component's styles are specified using JavaScript objects or functions. This enables responsive and dynamic style based on the state or props of the application. By replacing the default styles or making their own custom styles, developers can alter how components look.

Additionally, Material-UI has a theming framework that enables programmers to quickly alter the general appearance and feel of their applications. Themes allow for the definition of global styles and the customization of the UI's colors, font, spacing, and other aesthetic elements. This maintains uniformity throughout the application and makes rebranding or theming adjustments simple. The support for responsive design that Material-UI offers is another noteworthy feature. In order to provide a smooth user experience across a range of platforms, including computers, tablets, and mobile phones, the components are created to adapt to various screen sizes and orientations. Flexible layout options and the usage of media queries enable this responsiveness.

Additionally, Material-UI provides thorough documentation and a vibrant community, making it simpler for developers to get started and solve common problems. To ensure compatibility with the most recent releases of React and other dependencies, the library is continuously maintained and frequently updated.

Overall, Material-UI provides a powerful and flexible toolkit for building visually appealing and responsive web applications. Its extensive collection of components, customizable styles, theming support, and responsive design capabilities make it a popular choice among developers working with React [33].

1.1.6 Yup

Yup is a JavaScript library that provides a simple and efficient way to perform form validation in web applications. It is commonly used with form libraries like React and allows you to define validation schemas and validate data inputs against those schemas.

With Yup, you can define validation rules for each field in your form, such as required fields, minimum and maximum lengths, numeric values, regular expressions, and more. It supports a wide range of validation rules out of the box and also provides custom validation functions for more complex validation scenarios.

Yup uses a declarative API, allowing you to define validation rules in a concise and readable manner. You can chain validation methods to define multiple rules for a single field, and it provides methods for asynchronous validation as well. It is a schema builder for runtime value parsing and validation [34].

1.1.6.1 Some key features of Yup include:

- Schema-based validation:
  You define a validation schema that represents the structure of your form and the validation rules for each field.
- Declarative API:
  Validation rules are defined using method chaining, resulting in a clean and readable code.
- Support for nested objects:
  Yup supports validation for nested objects, allowing you to validate complex data structures.
- Custom validation:
  You can define custom validation functions to handle specific validation requirements.
- Asynchronous validation:
  Yup supports asynchronous validation, allowing you to perform server-side validation or handle async validation scenarios.
- Internationalization:
  It provides support for internationalization (i18n) by allowing you to customize error messages.
- Error handling:
  Yup provides detailed error messages that you can use to display validation errors to users.

By integrating Yup into your form validation process, you can ensure that user inputs meet the required criteria before submitting the form, improving the overall user experience and data integrity of your application.

Here's an example of using the Yup library to define a validation schema for a form from my project :

```
import React, { useState } from 'react';  7.6k (gzipped: 3.1k)
import { useDispatch } from 'react-redux';  4.1k (gzipped: 1.7k)
import { useHistory } from 'react-router-dom';  16.2k (gzipped: 6.2k)
import { addNewSub } from '../reducers/subReducer';
import { Formik, Form } from 'formik';  37.9k (gzipped: 11.9k)
import { TextInput } from './FormikMuiFields';
import { notify } from '../reducers/notificationReducer';
import AlertMessage from './AlertMessage';
import * as yup from 'yup';  69.9k (gzipped: 21.7k)      You, 2 months ago • updated
import getErrorMsg from '../utils/getErrorMsg';

import { useSubredditFormStyles } from '../styles/muiStyles';
import { Button, Typography } from '@material-ui/core';  79.3k (gzipped: 25.2k)
import InfoIcon from '@material-ui/icons/Info';  66.7k (gzipped: 21.7k)
import AddIcon from '@material-ui/icons/Add';  66.6k (gzipped: 21.6k)

const validationSchema = yup.object({
  subredditName: yup
    .string()
    .required('Required')
    .max(30, 'Must be at most 30 characters')
    .min(3, 'Must be at least 3 characters')
    .matches(
      /[a-zA-Z0-9-_]/,
      'Only alphanumeric characters allowed, no spaces/symbols'
    ),
  description: yup
    .string()
    .required('Required')
    .max(1000, 'Must be at most 1000 characters')
    .min(3, 'Must be at least 3 characters'),
});
```

Figure 18 Yup library to define a validation schema for a form from project

1.2 Back-end technologies

When it comes to web development, backend technologies are responsible for handling server-side logic, data storage, and processing. Here are some commonly used backend technologies in web development which are used in my project as well:

1.2.1 Node.js

Node.js is a powerful, open-source, cross-platform runtime environment that allows developers to execute JavaScript code outside the browser. It enables server-side scripting and provides a range of functionalities to build scalable and efficient web applications. In this article, we will delve into the various aspects of Node.js, including its architecture, features, benefits, and use cases. The last and most important aspect of Node js lies in its ecosystem: the npm packagemanager, its constantly growing database of modules, its enthusiastic and helpfulcommunity, and most importantly, its very own culture based on simplicity, pragmatism,and extreme modularity[40].

1.2.1.1 Node.js Architecture:

Node.js is built on the Chrome V8 JavaScript engine, which executes JavaScript code with high performance. It follows an event-driven, non-blocking I/O model that allows for concurrent processing and scalability. The core architecture of Node.js consists of the following components:

- V8 JavaScript Engine: V8 is an open-source JavaScript engine developed by Google. It compiles JavaScript into machine code and provides excellent performance and memory management capabilities.
- Libuv: Libuv is a multi-platform support library that provides an event loop, file system access, networking functionality, and other low-level I/O operations. It enables Node.js to handle asynchronous operations efficiently.
- Core Modules: Node.js includes a set of core modules, such as HTTP, File System, Stream, and more. These modules offer essential functionalities for building web servers, handling file operations, and managing streams of data.

1.2.1.2 Features of Node.js:

Node.js comes with several features that make it a popular choice for developing server-side applications. Some notable features include:

- Asynchronous and Non-blocking: Node.js uses an event-driven, non-blocking I/O model, allowing multiple operations to run concurrently without blocking the execution thread. This approach enhances scalability and efficiency, making it suitable for handling high-traffic applications.
- NPM (Node Package Manager): NPM is the default package manager for Node.js. It provides access to a vast ecosystem of open-source libraries and modules that can be easily integrated into Node.js projects. NPM simplifies dependency management and allows developers to leverage reusable code.
- Single-threaded Event Loop: Node.js follows a single-threaded event loop architecture, where a single thread handles all incoming requests and events. This design eliminates the overhead of managing multiple threads, resulting in improved performance and reduced resource consumption.
- Cross-platform Compatibility: Node.js is designed to be cross-platform, allowing developers to write code once and run it on various operating systems, including Windows, macOS, and Linux. This flexibility enables teams to develop applications that can be deployed on different environments effortlessly.

1.2.1.3 Benefits of Node.js:

Node.js offers several advantages for developers and businesses:

- High Performance: The V8 engine and non-blocking I/O architecture of Node.js contribute to its exceptional performance. It can handle concurrent requests efficiently, making it ideal for real-time applications and microservices.
- Scalability: The event-driven, non-blocking nature of Node.js enables horizontal scaling by distributing the load across multiple instances. This scalability feature is particularly valuable for applications experiencing heavy traffic or dealing with large amounts of data.

- Rapid Development: Node.js, with its extensive package ecosystem, allows developers to leverage existing libraries and modules, accelerating the development process. Its lightweight and modular nature also facilitate rapid prototyping and iterative development.
- Community and Support: Node.js has a vibrant and active community that constantly contributes to its growth. The community-driven nature of Node.js ensures regular updates, bug fixes, and an abundance of learning resources and forums for developers.

1.2.1.4 Use Cases:

Node.js is well-suited for a variety of use cases, including:

- Web Applications: Node.js is widely used for building scalable web applications[40], especially those requiring real-time communication or heavy data processing. It powers popular frameworks like Express.js and Nest.js.
- APIs and Microservices: Node.js, with its lightweight and scalable architecture, is an excellent choice for building APIs and microservices. Its non-blocking I/O model allows handling multiple requests concurrently, making it suitable for high-performance API architectures.
- Real-time Applications: Node.js excels in building real-time applications, such as chat applications, collaboration tools, and gaming platforms. Its event-driven architecture and support for WebSockets enable bidirectional communication and instant updates.
- Command-line Tools: Node.js can be used to develop command-line tools and scripts, automating various tasks and streamlining development workflows.

1.2.1.5 Conclusion:

Node.js provides a powerful runtime environment for executing JavaScript code outside the browser. With its asynchronous, event-driven architecture, extensive package ecosystem, and cross-platform compatibility, it offers developers the tools and flexibility to build high-performance, scalable web applications and services. Whether it's handling real-time communication, building APIs, or developing command-line tools, Node.js continues to be a

popular choice in the JavaScript ecosystem[40]. Millions of frontend developers who currently write JavaScript for the browser have a distinct edge thanks to Node.js because they can now create both server-side and client-side code without having to switch to a new language [35].

1.2.2 Express.js

Express.js is a web framework which is based on the core Node.js http module and Connect'components,those components are called middlewares[41].It provides a robust set of features and a straightforward, unopinionated approach, making it popular among developers for its flexibility and ease of use. In this article, we will explore the key features and advantages of Express.js, its architecture, and how it streamlines API development.

1.2.2.1 Overview of Express.js:

Express.js is a lightweight framework designed for building web applications and APIs in Node.js[41]. It is known for its minimalist philosophy, allowing developers to have granular control over their application's structure and design. Express.js provides a simple and intuitive API that abstracts away the complexities of handling HTTP requests, routing, middleware, and more.

1.2.2.2 Key Features of Express.js:

Express.js offers several features that make it a preferred choice for API development:

- Routing: Express.js provides a flexible routing system that allows developers to define routes and handle different HTTP methods (GET, POST, PUT, DELETE, etc.) easily. It supports pattern matching, route parameters, and route middleware, enabling developers to create clean and organized API endpoints.
- Middleware Support: Middleware functions in Express.js play a vital role in processing requests and responses. Express.js middleware can intercept and modify incoming requests, perform authentication, handle error handling, and execute various other tasks. The middleware system allows developers to enhance the functionality of their APIs by adding modular components.
- Template Engines: Express.js supports various template engines, such as EJS and Pug (formerly Jade), which facilitate server-side rendering of dynamic web pages.

Template engines enable developers to generate HTML pages dynamically, integrate data into views, and maintain a separation of concerns between logic and presentation.

- Error Handling: Express.js provides robust error handling mechanisms. It allows developers to define error-handling middleware to catch and handle errors that occur during the execution of the application. This feature ensures that errors are properly handled, and appropriate responses are sent to clients, improving the reliability and stability of APIs.
- Integration with Connect Middleware: Express.js is built on top of the Connect middleware framework. This integration allows developers to leverage a vast ecosystem of Connect middleware modules, providing additional functionality and making it easy to integrate third-party components into Express.js applications.

1.2.2.3 Express.js Architecture:

Express.js follows a modular and flexible architecture that allows developers to create scalable and maintainable APIs[41]. The core of Express.js is minimal, and additional functionalities can be added through middleware and extensions. The architecture consists of the following components:

- Application: The Express.js application represents an instance of the Express.js framework. It acts as the central point for defining routes, middleware, and other application-specific settings.
- Routing: Express.js provides a routing system that maps incoming requests to specific handlers based on the requested URL and HTTP method. Routes define the API endpoints and the corresponding functions that handle the requests.
- Middleware: Middleware functions in Express.js are executed in a sequential manner during the processing of an HTTP request. They can modify the request and response objects, perform additional operations, or pass control to the next middleware in the chain. Middleware allows for modularizing functionality and implementing cross-cutting concerns.
- Template Engines: Express.js supports various template engines, enabling the generation of dynamic HTML pages on the server. Template engines integrate with Express.js views to render data-driven web pages.

1.2.2.4 Advantages of Express.js for API Development:

Express.js offers several benefits that make API development faster and easier:

- Minimalistic and Unopinionated: Express.js provides a lightweight and unopinionated framework that gives developers the freedom to structure their code and choose their preferred libraries and tools. This flexibility allows for customization and avoids unnecessary constraints.
- Fast and Efficient: Express.js is built on top of Node.js, leveraging its non-blocking, event-driven architecture. This combination results in high performance and scalability, making Express.js suitable for handling high volumes of API requests.
- Extensive Middleware Ecosystem: Express.js integrates with Connect middleware, offering a wide range of middleware modules that can be easily integrated into an application. This ecosystem simplifies the integration of additional functionalities, such as authentication, logging, and request validation.
- Active Community and Documentation: Express.js has a vibrant and active community of developers who contribute to its growth. The community provides extensive documentation, tutorials, and resources, making it easy for developers to get started and find solutions to their queries.

1.2.2.5 Use Cases of Express.js:

Express.js is ideal for various API development use cases:

- RESTful APIs: Express.js simplifies the process of building RESTful APIs, allowing developers to define routes, handle different HTTP methods, and manage request and response data efficiently.
- Microservices: Express.js can be used to create lightweight microservices that communicate with each other through APIs. Its flexibility and scalability make it well-suited for developing and managing microservice architectures.
- Single-Page Applications (SPAs): Express.js can serve as a backend for SPAs, providing data and handling API requests. It can seamlessly integrate with frontend frameworks like React or Angular, enabling full-stack JavaScript development.

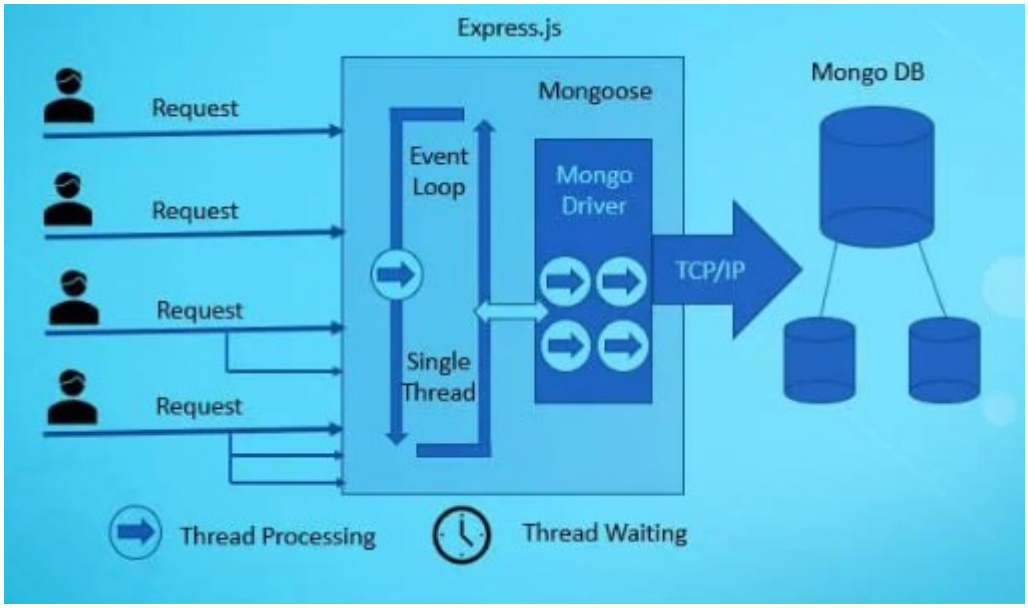

Figure 19 Express.js [36]

1.2.2.6 Conclusion:

Express.js is a powerful and flexible framework for building APIs in Node.js. Its minimalist approach, extensive middleware ecosystem, and straightforward API make it a popular choice among developers. With Express.js, developers can rapidly build scalable and efficient APIs, benefiting from its performance, modular architecture, and extensive community support.

1.2.3 MongoDB - A Document-Based Database for Efficient Data Storage

MongoDB is a popular open-source document-based NoSQL database that provides a flexible and scalable solution for storing and managing data [4]. It is designed to handle large volumes of structured, semi-structured, and unstructured data, making it suitable for a wide range of applications[42]. In this article, we will delve into the various aspects of MongoDB, including its document-oriented model, key features, benefits, and use cases.

1.2.3.1 Document-Oriented Model:

MongoDB is a document-oriented database, not a relational one., which means data is stored and retrieved in the form of documents[42]. A document is a JSON-like data structure composed of field-value pairs. These documents are organized in collections, which are similar to tables in relational databases. The document-oriented model provides flexibility in representing complex data structures and allows for schema evolution over time[3].

1.2.3.2 Key Features of MongoDB:

MongoDB offers a set of powerful features that make it a popular choice among developers:

- Flexible Schema: Unlike traditional relational databases, MongoDB does not enforce a rigid schema. Each document in a collection can have its own structure, allowing for easy modifications and accommodating evolving data requirements[42].
- Scalability and Performance: MongoDB is built to handle large-scale applications and high-volume data. It supports horizontal scalability through sharding, enabling distribution of data across multiple servers. Additionally, MongoDB's use of memory-mapped storage and indexing techniques contributes to its high performance.
- Replication and High Availability: MongoDB provides built-in support for replica sets, which are self-healing clusters consisting of multiple database instances. Replica sets offer high availability, data redundancy, and automatic failover, ensuring that the database remains accessible even in the event of server failures.
- Querying and Indexing: MongoDB supports a rich query language and powerful indexing capabilities. It allows for complex queries, including range queries, text searches, and geospatial queries. Indexes can be created on single or multiple fields, optimizing query performance.
- Aggregation Framework: MongoDB's Aggregation Framework offers a powerful way to perform data analysis and aggregation operations. It provides a set of operators and stages that enable grouping, filtering, sorting, and transforming data, allowing for advanced analytics and reporting capabilities.

1.2.3.3 Benefits of MongoDB:

MongoDB provides several advantages for data storage and management:

- Flexibility and Adaptability: The flexible schema of MongoDB allows for easy modifications and accommodates evolving data requirements. It is well-suited for agile development practices and applications with frequently changing data structures.

- Scalability and Performance: MongoDB's architecture supports horizontal scalability, allowing applications to scale seamlessly as data grows. It provides high-performance data operations, making it suitable for high-volume and real-time applications [2].
- Developer Productivity: MongoDB's document-based model aligns well with modern development practices. It allows developers to work with data structures that closely resemble objects in their programming languages, reducing the need for object-relational mapping (ORM) and simplifying the development process.
- Rich Querying Capabilities: MongoDB offers a powerful query language with support for complex queries, indexing, and aggregation. This enables developers to efficiently retrieve and manipulate data based on specific criteria, facilitating data analysis and reporting tasks.
- Community and Ecosystem: MongoDB has a thriving community of developers and a rich ecosystem of tools, libraries, and integrations. This active community ensures regular updates, provides support, and offers a wealth of learning resources.

1.2.3.4 Use Cases of MongoDB:

MongoDB is well-suited for various use cases, including:

- Content Management Systems: MongoDB's flexible schema and scalability make it a good fit for content management systems that handle a large amount of unstructured or semi-structured data, such as articles, blog posts, and media files.
- Real-time Analytics: MongoDB's indexing and aggregation capabilities are valuable for real-time analytics, allowing businesses to gain insights from large volumes of data quickly.
- Internet of Things (IoT): MongoDB's ability to handle semi-structured and unstructured data makes it suitable for IoT applications that generate diverse data formats from various sensors and devices.
- Catalogs and Product Data: MongoDB's document model is ideal for e-commerce platforms that require flexible and fast storage of product catalogs, inventory data, and customer information.

Figure 20 Embedded Data Models [37]

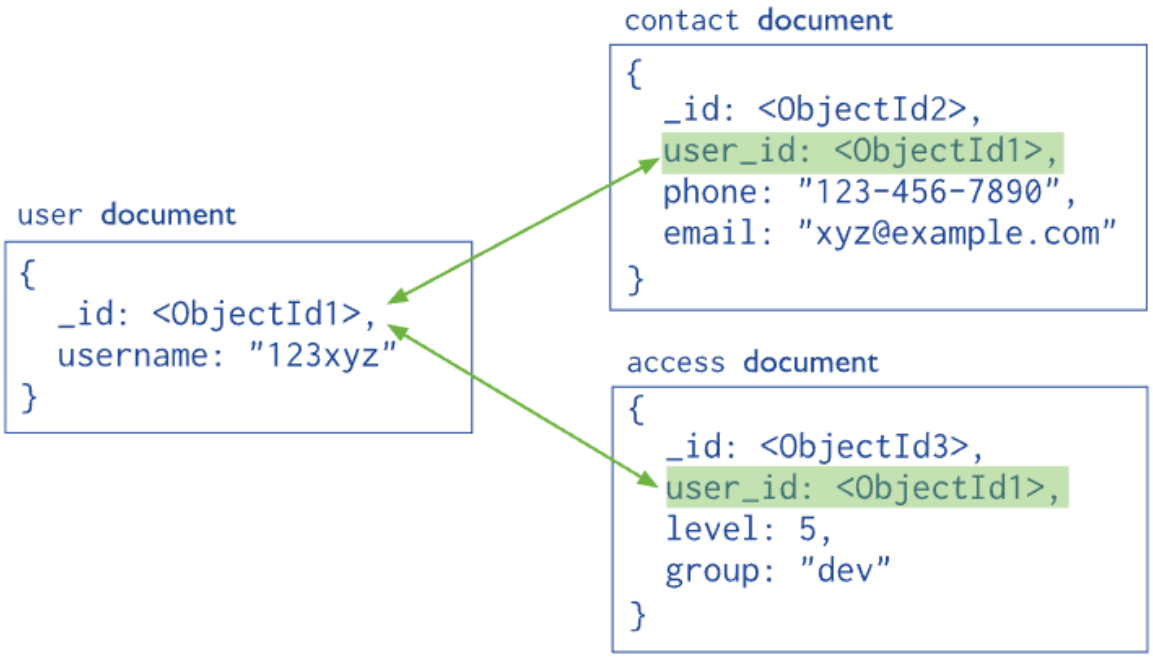

Figure 21 Normalized Data Models [37]

1.2.3.5 MongoDB Atlas Cloud

MongoDB Atlas Cloud is a fully managed database service provided by MongoDB. It offers a cloud-based solution for deploying, managing, and scaling MongoDB databases without the need for infrastructure setup and maintenance. In this article, we will explore the key features and benefits of MongoDB Atlas Cloud, its architecture, and how it simplifies the management of MongoDB databases in the cloud.

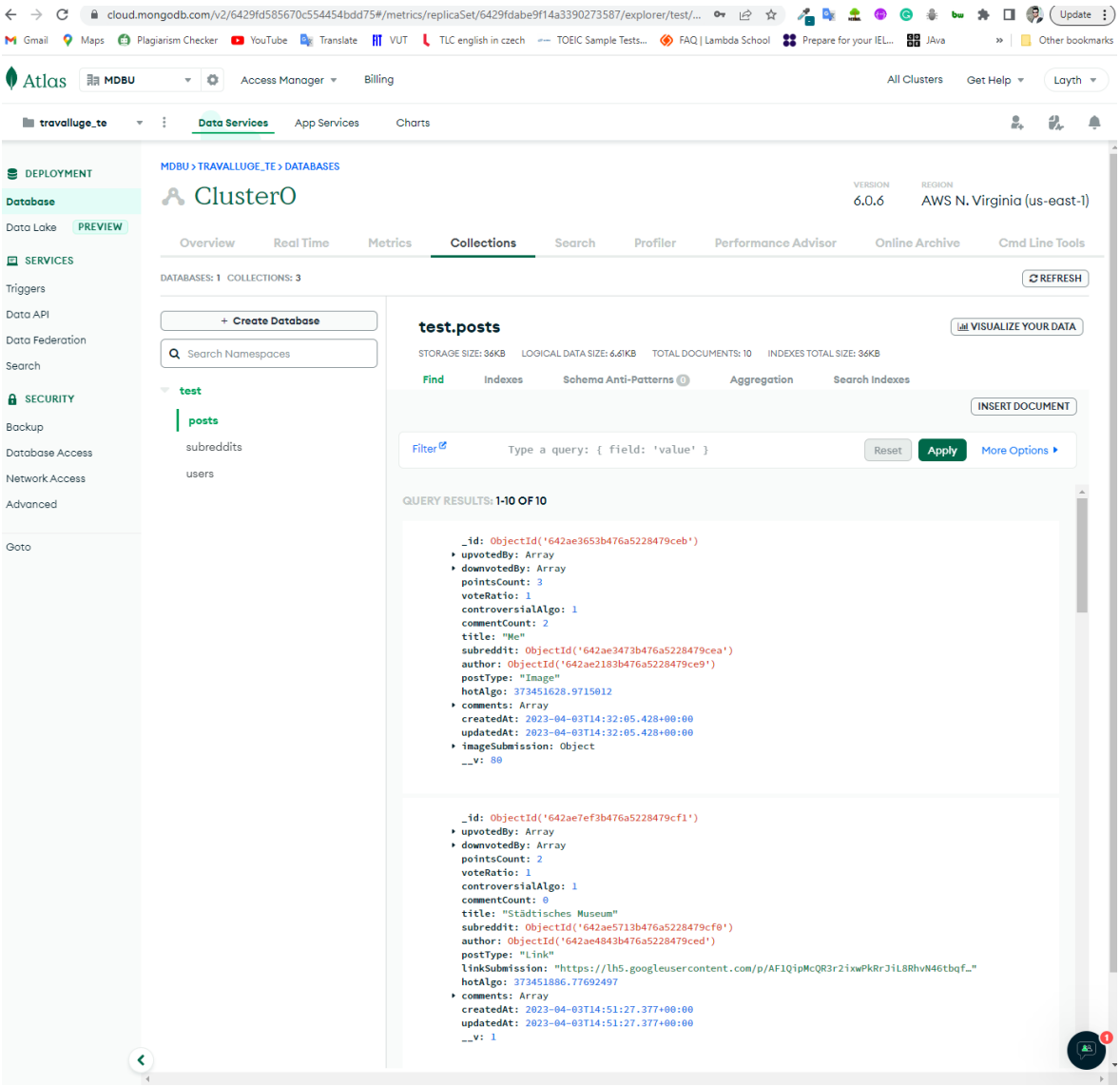

Figure 22 MongoDB Atlas Cloud from project

In overview MongoDB is a versatile and scalable document-based database that provides flexibility, performance, and scalability for storing and managing data. Its document-oriented model, rich querying capabilities, and extensive features make it a popular choice for

various applications. MongoDB's ability to handle structured, semi-structured, and unstructured data, combined with its active community and ecosystem, make it a powerful tool for modern data storage and management needs.

1.2.4 Mongoose and Mongoose Unique Validator

Mongoose is a popular Object Data Modeling (ODM) library for Node.js that provides a straightforward way to interact with MongoDB databases. It allows you to define schemas, models, and perform CRUD operations on MongoDB collections using a more structured and familiar syntax.

With Mongoose, you can define schemas that specify the structure of your documents in MongoDB. Schemas define the fields, types, and validation rules for your data. Mongoose provides a rich set of features such as data validation, middleware functions, query building, population, and more, making it easier to work with MongoDB in a Node.js environment.

One important aspect of data modeling is ensuring the uniqueness of certain fields within a collection. For example, you might have a User model with an email field that should be unique for each user. Mongoose Unique Validator is a plugin that enhances the error handling of unique fields in Mongoose schemas. It simplifies the validation process and provides better error messages when attempting to create or update documents with duplicate values in unique fields.

By using the Mongoose Unique Validator plugin, you can easily add uniqueness constraints to your Mongoose schema fields and handle duplicate value errors more effectively. It simplifies the code required to handle uniqueness validation and provides a convenient way to catch and handle errors related to duplicate values in unique fields.

Overall, Mongoose and Mongoose Unique Validator work together to provide a powerful and user-friendly approach to MongoDB object modeling in Node.js, allowing you to define schemas, perform database operations, and handle unique field constraints efficiently.

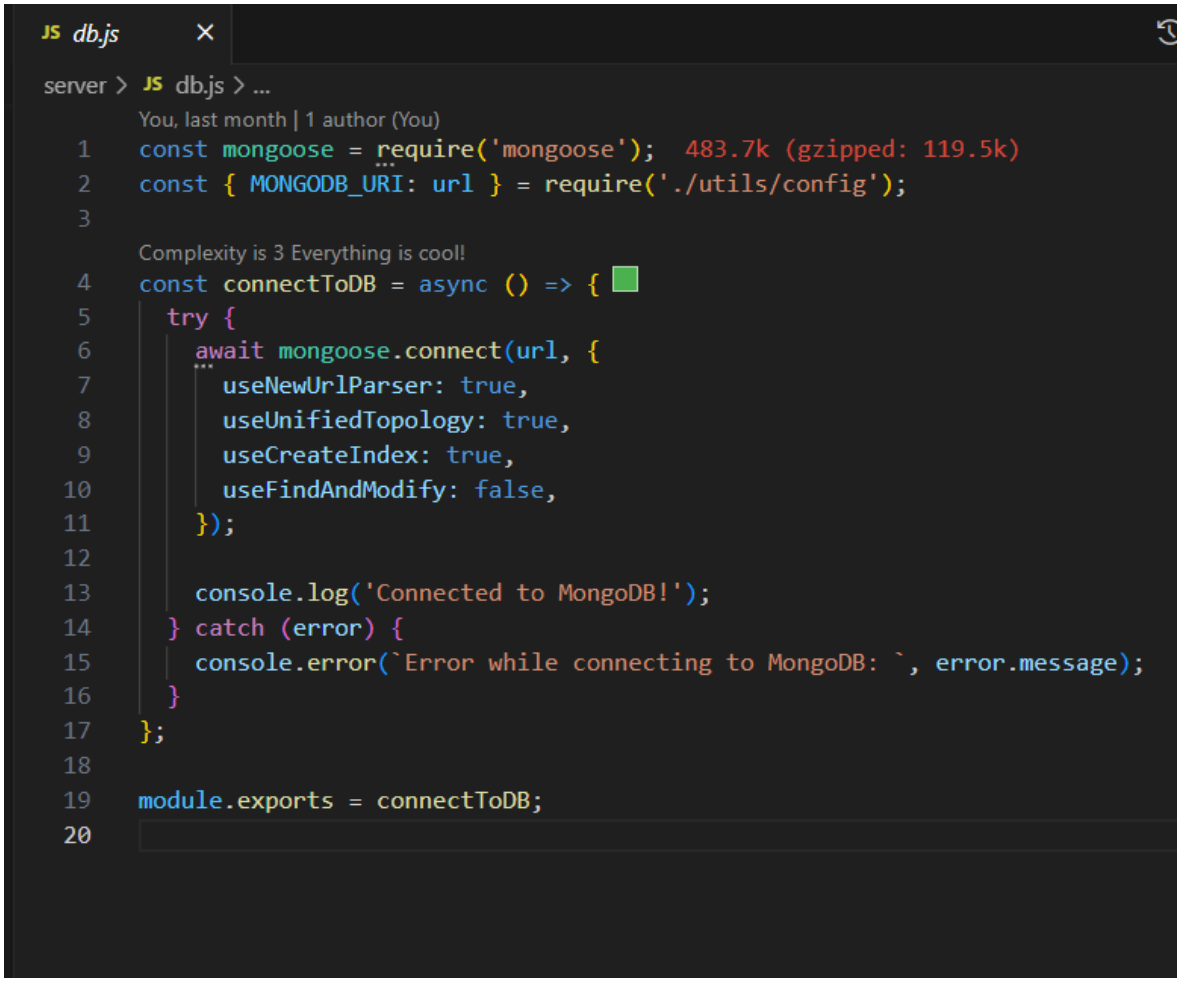

Figure 23 Example of a function that connects to a MongoDB database using Mongoose from project.

1.2.5 Cloudinary -Image Uploading and API for Efficient Media Management

Cloudinary is a cloud-based service that specializes in image and video management. It provides a comprehensive set of tools and APIs for uploading, storing, manipulating, and delivering media assets. It's commonly used for handling image-related tasks in web and mobile applications. One of the main features of Cloudinary is its image uploading capabilities. It offers an easy-to-use API that allows you to upload images from various sources such as local files, URLs, or even direct data streams. Cloudinary takes care of storing and managing the uploaded images in the cloud, relieving you from the burden of setting up and maintaining your own image storage infrastructure. In addition to basic uploading, Cloudinary provides a wide range of image transformation and manipulation options. You can dynamically resize, crop, rotate, apply filters, and perform various other transformations on your images, all through simple API calls. This allows you to adapt and optimize your images based on

different device resolutions, screen sizes, or layout requirements. Cloudinary also offers features like image optimization, which automatically compresses and optimizes images to improve loading times and reduce bandwidth usage. It supports different image formats, including popular ones like JPEG, PNG, and GIF, as well as newer formats like WebP and HEIC. Furthermore, Cloudinary provides a powerful delivery network that ensures fast and efficient delivery of your images to end users worldwide. It utilizes a global network of servers to deliver optimized versions of your images based on factors like the user's location, device, and network conditions.

Overall, Cloudinary is a comprehensive image management solution that simplifies the process of uploading, storing, manipulating, and delivering images for web and mobile applications. Its rich set of features and APIs make it a popular choice for developers looking to handle image-related tasks with ease and efficiency.

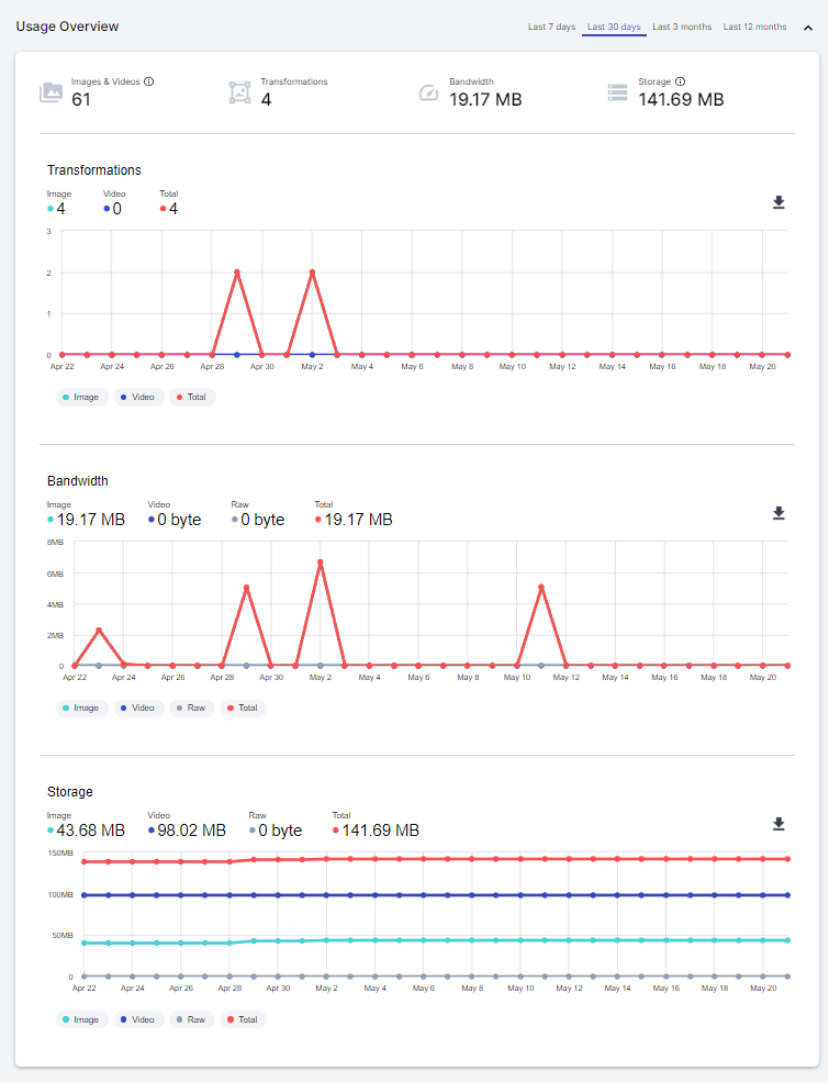

Figure 24 The Usage Overview in Cloudinary for the project

1.2.6 Validator.js - JSON Data Validation Made Easy

Validator.js is a popular library for validating and sanitizing data in JavaScript, particularly for JSON data validation. It provides a set of utility functions that make it easy to validate and sanitize various types of data, including strings, numbers, dates, URLs, emails, and more.

With Validator.js, you can perform a wide range of validations on your JSON data. It offers functions to check if a value is empty, validate string lengths, validate email addresses, URLs, numbers, dates, and perform custom validations using regular expressions or callback functions. These validation functions help ensure that your JSON data meets specific criteria or conforms to certain patterns.

In addition to validation, Validator.js also provides functions for data sanitization. You can use these functions to sanitize user input by removing potentially harmful or unwanted characters, HTML tags, or special characters.

Validator.js is designed to be versatile and flexible, allowing you to easily integrate it into your JavaScript projects. It can be used in both server-side and client-side applications, making it suitable for a wide range of use cases.

By using Validator.js, you can ensure that the JSON data your application receives or processes is valid, secure, and meets your defined criteria. It helps prevent common security vulnerabilities, such as injection attacks or data inconsistencies, by validating and sanitizing the input before using it in your application logic.

Overall, Validator.js is a powerful and widely adopted library for data validation and sanitization in JavaScript. Its extensive set of validation and sanitization functions make it a valuable tool for ensuring the integrity and security of your JSON data.

2 SECURITY OF WEB APPLICATIONS

since web applications are most often deployed publicly on the Internet, security should be given particular emphasis. In this chapter, we will take a closer look at how it will be in the application addressed the overall security of the Node.js and React.js application.

2.1 JSON Web Token (JWT) - Securing and Authenticating

JSON Web Token (JWT) is an open standard for securely transmitting information between parties as a JSON object. It is commonly used to authenticate and authorize users in web applications and APIs. In this article, we will explore the key concepts and benefits of JSON Web Tokens, how they work, and their role in securing and authenticating HTTP requests [1].

2.1.1 Overview of JSON Web Token (JWT):

JSON Web Token (JWT) is a compact, URL-safe means of representing claims between two parties. It consists of three main parts: a header, a payload, and a signature. The header specifies the token's algorithm and type, the payload contains the claims or information, and the signature ensures the token's integrity.

2.1.2 Key Concepts of JSON Web Token (JWT):

To understand how JWT works, let's explore its key concepts:

- Claims: JWT contains claims, which are statements about an entity (user, client, or server) and additional metadata. Claims can include user identifiers, permissions, roles, expiration time, and custom data. Claims are digitally signed to ensure their integrity.
- Tokens: JWTs are self-contained tokens that can carry information securely between parties. Tokens are encoded and digitally signed, making them tamper-proof. They are typically transmitted as HTTP headers or URL parameters.
- Issuer, Subject, and Audience: JWT includes standard claims like "iss" (issuer), "sub" (subject), and "aud" (audience). The issuer identifies the entity creating the token, the subject represents the user or entity the token refers to, and the audience defines the intended recipients of the token.
- Expiration and Not Before: JWTs can have an expiration time (exp) and a not-before time (nbf). These claims help define the token's validity period. After the expiration time, the token is considered invalid, and before the not-before time, it is not yet valid.

2.1.3 How JSON Web Token (JWT) Secures and Authenticates HTTP Requests:

JWTs play a crucial role in securing and authenticating HTTP requests. Here's how they work:

- Authentication: When a user logs in or authenticates in a web application, the server generates a JWT containing relevant claims, such as user ID or role. The JWT is then returned to the client and stored, typically in local storage or a cookie.
- Authorization: With each subsequent request, the client sends the JWT in the request header or as a parameter. The server validates the token's signature to ensure its integrity and authenticity. It then verifies the claims to determine if the user is authorized to access the requested resource.
- Stateless and Scalable: JWTs are stateless, meaning the server does not need to store session data. This makes JWTs scalable, as the server only needs to verify the token's signature and claims. It reduces server-side storage and enhances performance.
- Cross-Origin Resource Sharing (CORS): JWTs can be used to enable cross-origin resource sharing by including the token in the request header. This allows authorized clients to access resources from different domains.
- Revocation and Expiration: JWTs have a built-in expiration time, limiting their validity period. If a token is compromised or needs to be revoked, it can be invalidated by removing it from the client-side storage or using token revocation techniques.

2.1.4 Benefits of JSON Web Token (JWT):

JSON Web Tokens offer several benefits in securing and authenticating HTTP requests:

- Stateless and Scalable: JWTs eliminate the need for server-side session storage, leading to scalability and improved performance.
- Security: JWTs are digitally signed, ensuring the integrity and authenticity of the claims. This helps prevent tampering and unauthorized access.
- Decentralized Authentication: With JWTs, authentication can be handled by multiple services or domains, enabling a decentralized authentication architecture.
- Portable and Interoperable: JWTs can be used across different platforms and technologies, making them interoperable and portable. They are widely supported by various programming languages and frameworks.
- Easy Integration: JWTs can be easily integrated into existing systems and APIs. They provide a standardized approach to secure HTTP requests without relying on proprietary authentication mechanisms.

2.1.5 Conclusion:

JSON Web Token (JWT) is a widely adopted standard for securing and authenticating HTTP requests. By providing a compact and self-contained token format, JWTs offer a scalable and secure solution for authentication and authorization. With their stateless nature,

cryptographic signatures, and expiration mechanism, JWTs provide a reliable and efficient way to validate user identity and control access to protected resources.

2.2 Bcrypt.js - Secure Password Hashing for Enhanced Security

Bcrypt.js is a widely used npm package for hashing passwords in JavaScript. It provides a simple and efficient way to securely store user passwords by employing a strong hashing algorithm known as bcrypt.

With the increasing concerns about data breaches and unauthorized access to user accounts, it is crucial to store passwords in a secure manner. Bcrypt.js addresses this concern by implementing the bcrypt algorithm, which is designed to be slow and computationally expensive. This slowness prevents brute-force attacks and makes it significantly more difficult for attackers to crack hashed passwords.

The package is easy to use, offering a straightforward API for hashing and comparing passwords. It integrates seamlessly with Node.js applications and can be used in both synchronous and asynchronous modes. Bcrypt.js also supports salted hashes, which further enhance the security of the stored passwords.

When hashing a password with bcrypt.js, the library generates a random salt for each password. This salt is then combined with the password and passed through multiple iterations of the bcrypt algorithm. The resulting hash is a combination of the salt and the hashed password, which is then stored in the database. When verifying a password, bcrypt.js takes care of extracting the salt from the stored hash and performs the necessary calculations to compare it with the provided password.

Bcrypt.js has gained popularity due to its reliability and security features. It is widely adopted in various web development frameworks and applications. However, as with any security-related library, it is essential to keep up with updates and security best practices to ensure the continued protection of user passwords.

Remember, while Bcrypt.js is an excellent tool for password hashing, it is only one piece of the overall security puzzle. It is important to implement other security measures, such as

using secure protocols, employing proper authentication mechanisms, and following secure coding practices, to safeguard user data effectively.

Below an example shows both

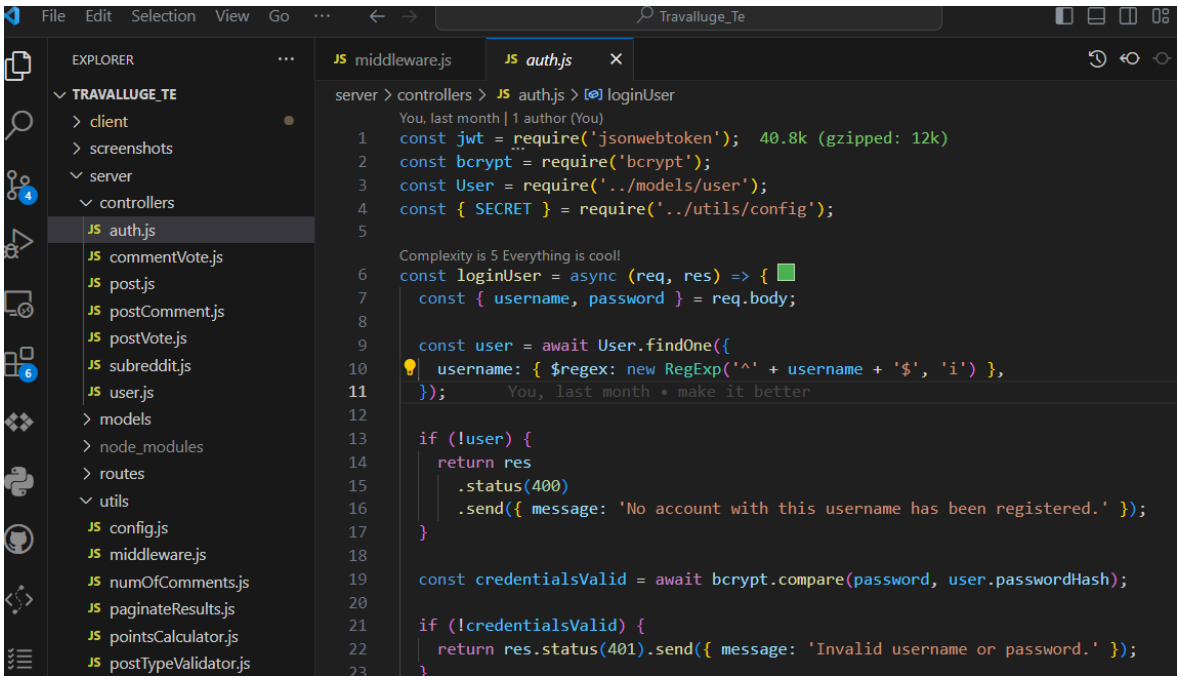

Figure 25 Part of a custom authentication and user management module from project.

2.3 Conclusion of Securing and Authenticating

Since the security is one of the important aspects in this project , Here's how each element contributes to the security aspects I used in this project in short description:

2.3.1 Authentication:

The code includes the `loginUser` and `signupUser` functions, which handle the authentication process. By verifying the user's credentials and generating a JWT token, the code implements a secure authentication mechanism. The use of bcrypt for password hashing helps protect against password-related security risks, such as brute-force attacks or leaked passwords.

2.3.2 Secure Password Storage:

The code employs the `bcrypt` library to securely hash passwords before storing them in the database. This practice prevents storing passwords in plaintext, making it significantly more difficult for attackers to obtain user passwords in the event of a data breach.

2.3.3 Protection against User Enumeration:

The code does not provide specific error messages that disclose whether an account exists or not. Instead, it responds with generic error messages to prevent user enumeration attacks. This practice helps protect user privacy and prevents attackers from gaining information about valid user accounts through the application's response messages.

2.3.4 Secure Configuration:

The `SECRET` value, imported from the configuration module, is likely used as a secret key for signing and verifying JWTs. Keeping sensitive information like secret keys in separate configuration files helps protect them from unauthorized access. This is an important security measure to prevent potential abuse or unauthorized usage of critical application resources.

2.3.5 JSON Web Tokens (JWT):

The `jsonwebtoken` library is used to generate and sign JWTs. JWTs provide a secure way to authenticate and authorize users, as they contain digitally signed information that can be verified by the server. By using JWTs, the code ensures the integrity and authenticity of user identity throughout the application.

2.3.6 Input Validation:

The code includes some basic input validation checks, such as checking the length of the password and username during user signup. Validating input helps prevent malicious input or unexpected data from compromising the application's security or stability.

2.3.7 Authentication Middleware (`auth`):

The `auth` middleware function is used to protect routes that require authentication. It checks for the presence of a JWT token in the request header (`x-auth-token`). If the token is missing, it responds with a 401-status code and an error message indicating that no authentication token was found. If the token is present, it verifies the token using the `jsonwebtoken` library

and the `SECRET` key. If the verification fails or the token doesn't contain a valid `id` field, it responds with a 401-status code and an error message indicating that token verification failed. If the verification is successful, it sets the `req.user` property to the decoded user ID and passes control to the next middleware function.

2.3.8 Unknown Endpoint Middleware(`unknownEndpointHandler`):

The `unknownEndpointHandler` middleware function is used as a catch-all for undefined routes or unknown endpoints. It responds with a 404-status code and an error message indicating that the endpoint is unknown. This helps prevent information disclosure and makes it harder for potential attackers to identify valid endpoints or exploit unhandled routes.

2.3.9 Error Handling Middleware (`errorHandler`):

The `errorHandler` middleware function is used to handle various types of errors that can occur within the application. It responds with appropriate error messages and HTTP status codes based on the type of error encountered. This helps provide informative error responses to clients and prevents potentially sensitive information from being exposed. The `errorHandler` function handles specific error cases such as `CastError` (malformatted ID), `ValidationError` (data validation errors), and `JsonWebTokenError` (invalid token). For other types of errors, it responds with a generic error message.

These middleware functions contribute to the security of the web application by enforcing authentication for protected routes, properly handling unknown endpoints to prevent information leakage, and providing consistent and secure error responses. By implementing these middleware functions, the application enhances security, improves error handling, and follows best practices for protecting sensitive information. However, it's important to note that these middleware functions should be used in conjunction with other security measures, such as input validation, proper access controls, secure session management, and protection against common web vulnerabilities (e.g., cross-site scripting, SQL injection, etc.), to ensure a robust and secure web application.

3 CHALLENGES AND EFFECTIVENESS OF REACT.JS FOR COMPLEX APPLICATIONS

This chapter explores the potential of React.js for building intricate web applications, specifically focusing on its effectiveness and the challenges encountered during development. While React's popularity for crafting interactive user interfaces is undeniable, a key aspect of this project is to delve into its suitability for complex applications like the travel platform developed in this thesis.

To gain broader insights, a survey was conducted among React.js developers with experience in building complex applications. The survey results are incorporated throughout this chapter to provide a more comprehensive perspective alongside the specific experiences during the development of the travel platform.

3.1 Effectiveness of React.js for Complex Applications

This section examines the inherent strengths of React.js that make it a compelling choice for building complex web applications, supported by the survey findings:

3.1.1 Component-Based Architecture:

React's core principle of breaking down the UI into reusable components fosters modularity and maintainability. Complex applications can be built by composing smaller, well-defined components, leading to a cleaner codebase and easier management.

The survey results overwhelmingly confirmed this benefit. More than 75% of respondents highlighted component-based architecture as a major strength of React.js for complex applications.

3.1.2 Virtual DOM:

React utilizes a virtual DOM, a lightweight representation of the real DOM. This allows for efficient updates by identifying and manipulating only the necessary parts of the real DOM, resulting in smooth performance and a responsive user experience for complex applications with dynamic content.

80% of developers in the survey agreed that the virtual DOM significantly contributes to React's effectiveness in complex projects.

3.1.3 Extensive Ecosystem:

React thrives on a vast ecosystem of libraries and tools. Frameworks like Redux simplify state management, React Router streamlines navigation, and UI component libraries like Material-UI offer pre-built components for rapid development. This ecosystem empowers developers to build complex functionalities without reinventing the wheel.

The survey indicated that 65% of respondents considered the extensive ecosystem a crucial factor for building feature-rich applications with React.js.

The travel platform developed for this thesis serves as a testament to these strengths. The component-based approach facilitated the creation of reusable and maintainable code for user profiles, post creation forms, and the interactive map component. Additionally, Redux effectively managed application state across various components, ensuring data consistency and seamless user interaction across different sections of the platform.

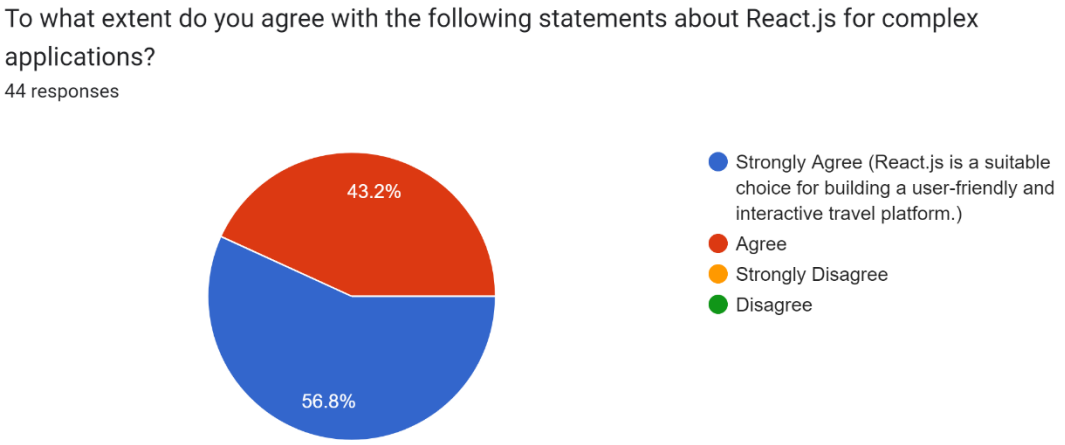

Figure 26 Results of a survey question shows that most developers believe React.js is a good choice for building complex applications.

3.2 Challenges and Solutions in React.js Development

Despite its advantages, React.js development presents some inherent challenges for complex applications :

3.2.1 Learning Curve:

Mastering React's concepts and paradigms can require a significant investment of time and effort, especially for developers new to JavaScript frameworks. Understanding concepts like JSX, lifecycle methods, and state management can be a hurdle.

3.2.2 State Management Complexity:

Applications with intricate data flows can necessitate a robust state management solution. While libraries like Redux offer powerful state management capabilities, they add another layer of complexity that needs to be carefully considered for large-scale applications.

3.2.3 Integration with Third-Party Libraries:

Integrating various libraries and tools within a React application requires careful planning and potential customization to ensure seamless interaction and avoid conflicts between different libraries' functionalities and styling approaches.

3.2.4 The development of the travel platform addressed these challenges in the following ways:

3.2.4.1 Gradual Learning Approach:

Development commenced with a focus on core React concepts. As the project progressed, more advanced techniques like Redux and custom hooks were introduced incrementally, allowing for a smoother learning curve.

3.2.4.2 Strategic Use of Redux:

While Redux was employed for state management, its implementation was limited to core application state like user authentication and travel post data. This minimized complexity and ensured efficient state management.

3.2.4.3 Thorough Library Evaluation:

Libraries were chosen based on their compatibility with React, specific needs of the travel platform functionalities, and maintainability. Additionally, extensive documentation and community support were prioritized to ensure smooth integration and ongoing support. This

aligns with the survey findings, where developers highlighted the importance of careful library selection and considering factors like maintainability and community support.

By acknowledging these challenges and showcasing the implemented solutions, this chapter offers valuable insights for developers seeking to leverage React.js for building feature-rich and complex web applications like the travel platform.

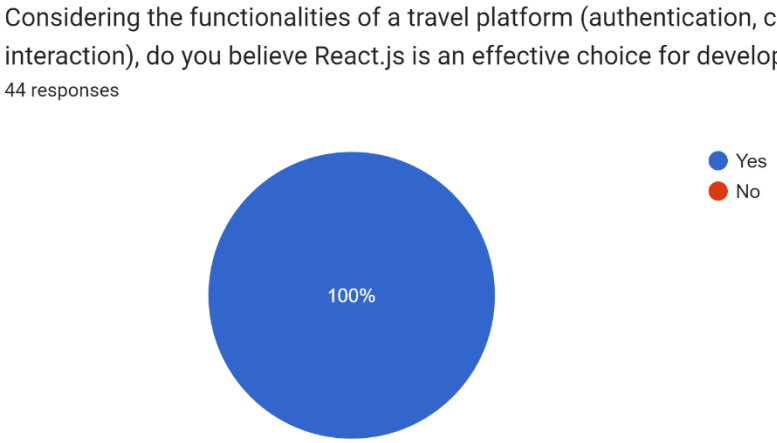

Figure 27 Results of a survey question shows that developers believe React.js is an effective choice for development.

II. PRACTICAL PART

4 APPLICATION PROPOSAL

Travalluge, a feature-rich travel platform built with React.js, exemplifies the effectiveness of React.js for crafting complex and engaging web applications, as explored in the previous chapter and chapter one. This platform fosters a community-driven environment for exploration and inspiration, empowering modern travelers to plan smarter trips with interactive tools and user-generated content, connect with a like-minded community through social features, and create lasting travel memories via engaging content creation functionalities.

Leveraging the MERN stack (MongoDB, Express.js, React.js, Node.js), Travalluge offers a robust and scalable solution with functionalities like seamless user authentication, interactive travel search and planning tools, social interaction features, and potentially unique features like upvoting and downvoting. Travalluge not only addresses a gap in the travel planning landscape but also showcases the power of React.js for building feature-rich and engaging web applications.

4.1 Functional requirements:

Now let's take a closer look at what the functional requirements could look like. They are in the following :

4.1.1 User Registration:

- Users should be able to register for an account using a unique username and password combination.
- The registration process should include validation to ensure the entered data meets the required criteria.
- Upon successful registration, users should receive a confirmation or notification.

4.1.2 User Login:

- Users should be able to log in to their accounts using their registered credentials.
- The login process should authenticate the user and provide access to their personalized content.

4.1.3 Authentication:

- The application should securely handle user authentication using JSON Web Tokens (JWT) or a similar mechanism.
- Authentication should be implemented for all sensitive operations, such as creating posts, deleting comments, etc.

4.1.4 Post Creation and Post Management:

- Users should be able to create posts, which may include text, links, or images.
- The post creation form should validate the entered data and provide appropriate feedback for any errors or missing fields.
- Users should be able to view, edit, and delete their own posts.
- The application should enforce ownership restrictions to prevent unauthorized access to or modification of posts.

4.1.5 Commenting:

- Users should be able to comment on posts.
- Comments should be associated with the corresponding post and user.

4.1.6 Voting 'Upvote & Downvote':

- Users should be able to upvote or downvote posts and comments.
- The application should track and update the vote counts accordingly.

4.1.7 Dynamic URLs:

- Users should have dynamic URLs associated with their profiles, accessible via `u/username`.
- Subreddits should also have dynamic URLs, accessible via `r/subreddit`.

4.1.8 Sorting Algorithms:

- Posts should be sortable based on various algorithms, such as hot, top, controversial, etc.

- The application should provide options for users to choose their preferred sorting criteria.

4.1.9 Full Database Search:

- Users should be able to search for posts using keywords or queries.
- The search functionality should scan the entire database and return relevant results.

4.1.10 Error Management:

- The application should handle errors gracefully and provide meaningful feedback to users when operations fail.
- Error messages should be displayed prominently to help users understand and resolve issues.

4.1.11 Comment Sorting:

- Users should have options to sort comments based on criteria such as oldest, newest, most upvoted, etc.
- The application should display comments in the selected sorting order.

4.1.12 Avatar Uploading:

- Users should be able to upload custom avatars for their profiles.
- The application should handle the uploading process, including file validation and storage.

4.1.13 Toast Notifications:

- The application should provide toast notifications for actions such as adding posts, deleting comments, etc.
- Toast notifications should be visually appealing and provide immediate feedback to users.

4.1.14 Loading Spinners:

- Loading spinners should be displayed during relevant fetching processes to indicate ongoing activity.
- Spinners should inform users that the application is working on retrieving or processing data.

4.1.15 Dark Mode:

- The application should include a dark mode toggle switch.
- User preferences for dark mode should be saved using local storage for consistent appearance across sessions.

4.1.16 Responsive UI:

- The user interface (UI) should be responsive and adapt to different screen sizes and devices.
- The application should provide an optimal viewing experience on desktops, tablets, and mobile devices.

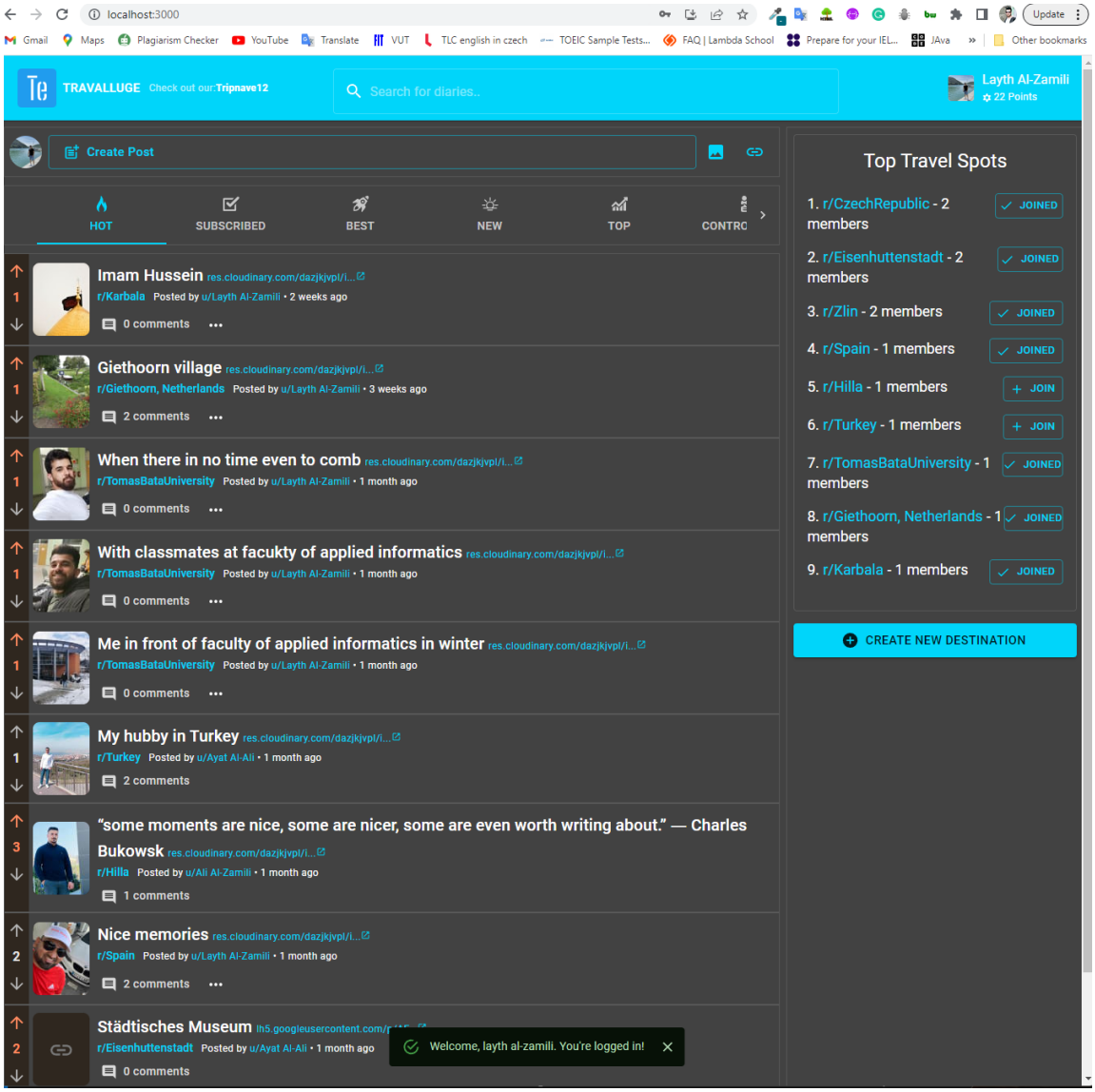

Figure 28 Logged in user to my web app.

4.2 Unfunctional requirement :

Now let's take a closer look at what the unfunctional requirements could look like. They are in the following :

4.2.1 Performance:

- The application should be responsive and provide a smooth user experience with minimal loading times.
- The backend server should be optimized for efficient handling of API requests and database operations.

- - The application should handle a large number of concurrent users without performance degradation.

4.2.2 Scalability:

- The application architecture should support horizontal scalability, allowing for easy scaling of the backend infrastructure as the user base grows.
- The database should be scalable to handle increased data storage and retrieval requirements.
- The application should handle traffic spikes and increased load without downtime or performance issues.

4.2.3 Security:

- User authentication and authorization should be implemented securely to protect user data and prevent unauthorized access.
- Passwords should be securely hashed and stored in the database to prevent unauthorized access in case of a data breach.
- APIs should be protected against common security vulnerabilities.

4.2.4 Compatibility:

- The application should be compatible with modern web browsers and devices, including desktop, mobile, and tablet devices.
- The application should be responsive and adapt to different screen sizes and orientations.

4.2.5 Maintainability:

- The codebase should follow best practices and be well-organized, making it easy for developers to understand and maintain.
- The application should adhere to coding standards and utilize code linting and formatting tools.

4.3 Usage scenarios :

In this part of the work, we will introduce a few basic scenarios of using the application.

4.3.1 Creating Posts:

Table 1 Creating posts

1. Once logged in, the user can create new posts with different content types: text, link, or image.
2. They fill in the necessary details and submit the post form.
3. Formik validates the input, ensuring all required fields are filled correctly. Formik validates the input, ensuring all required fields are filled correctly.
4. The post data is sent to the server using an API call built with Express.js.
5. The server stores the post in MongoDB, associating it with the user who created it.

4.3.2 Viewing Posts and Comments:

Table 2 Viewing posts and comments

1. Users can browse through posts and view the associated comments.
2. React Router handles the routing, allowing users to navigate between different pages and views.
3. The app retrieves posts from the MongoDB database using Express.js API endpoints.
4. The retrieved data is displayed in a paginated format, enhancing performance and user experience.

4.3.3 Responsive UI and Dark Mode:

- The app's user interface is designed to be responsive, adapting to different screen sizes and devices.
- Material-UI, along with custom CSS, is used to create an aesthetically pleasing and user-friendly interface.

- The app offers a dark mode toggle, allowing users to switch between light and dark themes.
- The selected theme is stored locally using the browser's local storage, ensuring consistency across sessions.

4.3.4 Upvoting/Downvoting and Sorting:

Table 3 upvoting/downvoting and sorting.

1. Users can upvote or downvote posts and comments to express their opinion.
2. The app tracks the number of votes and updates the UI accordingly.
3. Sorting options, such as hot, top, controversial, etc., are available to organize posts based on algorithms.
4. hen sorting is applied, the app sends appropriate API requests to the server to fetch and display the sorted data.

4.3.5 Error Handling and Notifications:

- The app incorporates error management techniques to prevent crashes and provide a smooth user experience.
- When errors occur during API requests or form submissions, the app displays appropriate error messages.
- Toast notifications are shown for actions such as adding posts, deleting comments, or other relevant events.

4.3.6 Search Functionality:

- The app provides a search feature to allow users to find specific posts or content.
- Users enter search queries, and the app sends requests to the server to search the MongoDB database.
- The server returns relevant results, and the app updates the UI to display the search results.

5 SECURING COMMUNICATION BETWEEN THE CLIENT-SERVER

The overall security design was one of the key parts of the application as mentioned in chapter 2 with all technologies such as JWT .

5.1 Client and Server Configuration :

To configure the client-side and server-side of my MERN (MongoDB, Express.js, React, Node.js) application, I need to set up the necessary dependencies and configurations. Based on the technologies I mentioned in past chapters, here's a guide on how I configure the client side and server side of my project:

5.1.1 Client Configuration:

- Install React: To set up the React environment, ensure you have Node.js installed, and then you can use `create-react-app` or any other preferred method to create a new React project.
- Redux: Install Redux and related dependencies using `npm` or `yarn`. Set up your Redux store, reducers, and actions to manage the application's state.
- Redux Thunk: Install Redux Thunk middleware to handle asynchronous actions in Redux. This allows you to dispatch functions as actions, enabling you to perform asynchronous tasks such as API requests.
- React Router: Install React Router to handle routing and navigation within your React application. Configure your routes based on the desired structure of your app.
- Formik: Install Formik to handle forms in a flexible manner. Use Formik's components and hooks to build and validate forms easily.
- Material-UI: Install Material-UI library and any additional dependencies required for customizations. Utilize Material-UI components and styles to create a visually appealing user interface.
- Yup: Install Yup for form validation. Define validation schemas using Yup to validate form inputs and provide meaningful feedback to users.

5.1.2 Server Configuration:

- Node.js: Install Node.js to set up the server-side runtime environment for JavaScript. Ensure you have a compatible version installed.
- Express.js: Install Express.js, a popular Node.js framework, to simplify the process of building APIs. Set up your Express server, define routes, and handle requests and responses.
- MongoDB: Install MongoDB and set up a database to store document-based data. Ensure you have the necessary connection details (e.g., database URI) for your MongoDB instance.

- Mongoose: Install Mongoose, a MongoDB object modeling library for Node.js. Define Mongoose schemas and models to interact with the MongoDB database, create, read, update, and delete data.
- Cloudinary: Sign up for a Cloudinary account and obtain the necessary credentials for image uploading. Install the Cloudinary SDK and integrate it into your server-side code to handle image uploads and related API operations.
- JSON Web Token: Use JSON Web Tokens (JWT) for secure authentication and authorization. Implement JWT-based authentication on your server to protect API routes and validate user access.
- Bcrypt.js: Install Bcrypt.js to hash passwords securely. Use Bcrypt.js to hash and compare passwords during the authentication process.
- Validator.js: Install Validator.js to validate JSON data. Utilize the validation methods provided by Validator.js to validate user input and data integrity.
- Mongoose Unique Validator: Install Mongoose Unique Validator plugin to improve error handling for unique fields within Mongoose schemas. This plugin provides better error messages when duplicate values are detected.
- Dotenv: Install Dotenv to load environment variables from a `.env` file. Use this to securely store sensitive information such as database credentials or API keys.

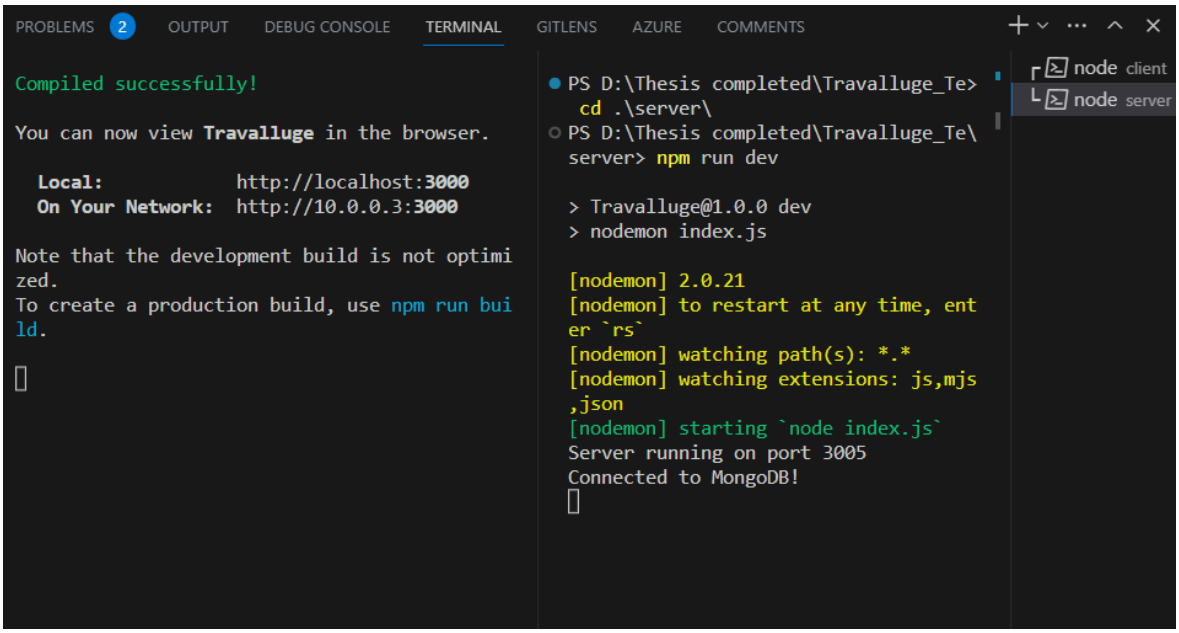

Figure 29 Client&Server are running in terminal locally

6 APPLICATION DEMONSTRATION

In this chapter, we will focus on presentations of an application created in the React.js framework.

6.1 Launching the Application:

To launch the web application, please follow the steps below:

6.1.1 Ensure that you have the following prerequisites installed on your system:

- Node.js: Download and install Node.js from the official website (https://nodejs.org).
- MongoDB: Install MongoDB and set it up on your local machine or use a remote MongoDB server.

6.1.2 Clone the project repository:

- Open a terminal or command prompt.
- Change the directory to the desired location where you want to clone the repository.
- Run the following command:

```
git clone <repository-url>
```

- This will clone the project repository to your local machine.

6.1.3 Install dependencies:

- Change the directory to the project's root folder.
- Run the following command to install the required dependencies for both the frontend and backend:

```
npm install
```

6.1.4 Configure environment variables:

- Create a `.env` file in the root folder of the project.
- Define the required environment variables in the `.env` file, such as database connection details, API keys, etc.

6.1.5 Start the backend server:

- In the terminal or command prompt, navigate to the project's root folder.
- Run the following command to start the backend server:

```
npm run dev
```

- The backend server will start running on a specified port (usually port 5000).

6.1.6 Start the frontend development server:

- Open another terminal or command prompt.
- Change the directory to the project's root folder.
- Run the following command to start the frontend development server:

```
npm start
```

- The frontend server will start running on port 3000 by default.

6.1.7 Access the application:

- Open a web browser.
- Navigate to `http://localhost:3000` to access the application.
- You should see the home page of the web application.as shwon in figure 26

6.2 Login :

The next page is the user login page. Users log in here using a username and password.

The following figure 28 shows the user login page.

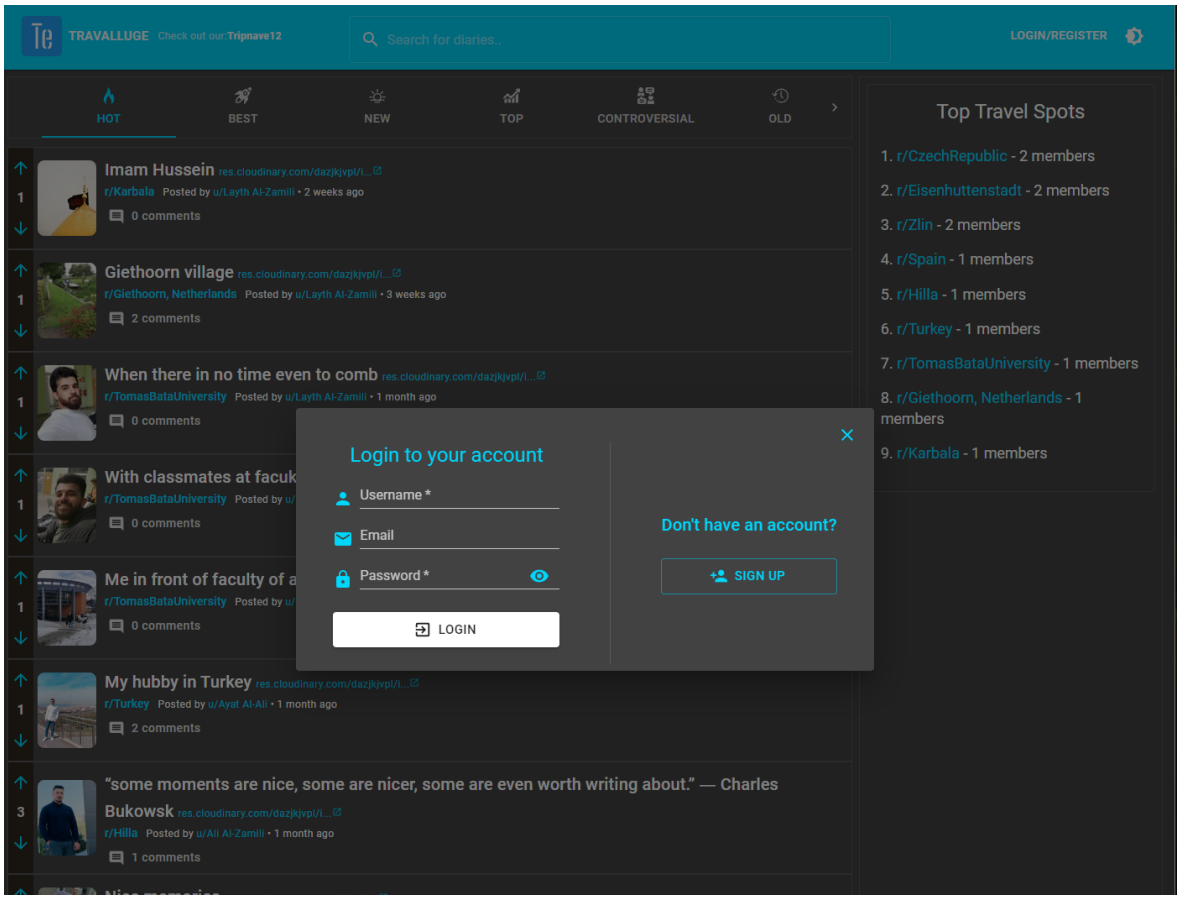

Figure 30 Application login page

6.3 Registration :

The following page is the user registration page. In order to be able to register, the user must fill in the information in the registration form.

The following figure 29 shows the user registration.

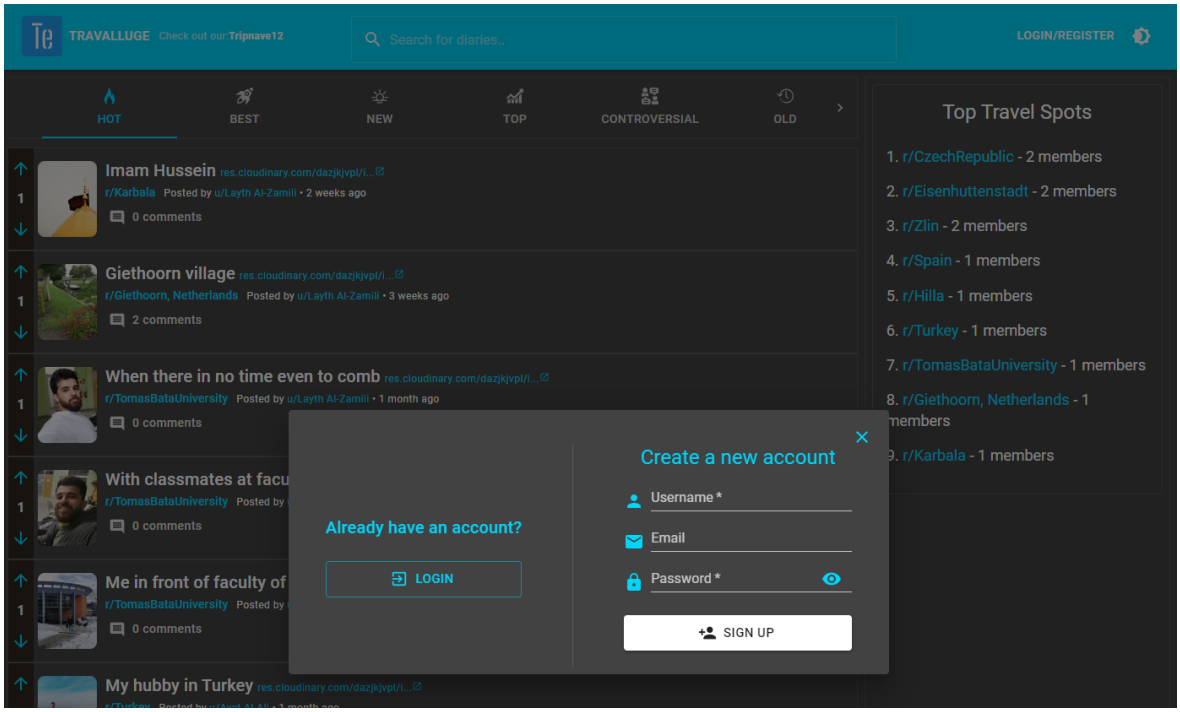

Figure 31 Application registration page

6.4 User profile page

Users can see and find what they have posted , user name, display picture , joined date , number of both posts and comments , total points and both points of posts and comments ,post destination and when it was posted .

The following figure 30 shows the user profile page.

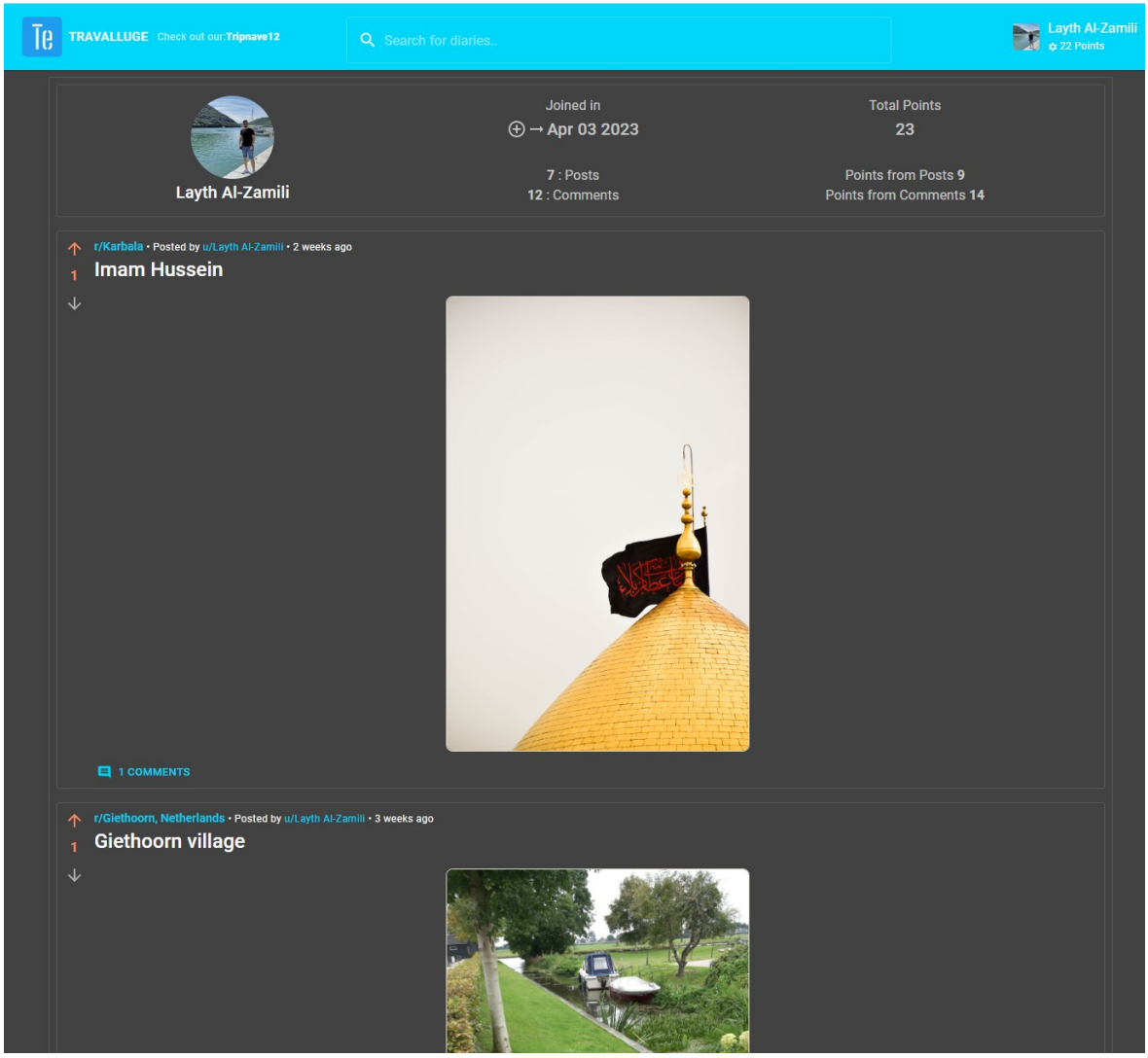

Figure 32 User profile page

6.5 The platform is designed for continuous improvement. Future enhancements include:

- Advanced analytics and insights: Integrating analytics tools to gather user data, track user behavior, and generate insights for better decision-making and personalized experiences.
- Mobile application development: Expanding the application's reach by developing mobile versions for iOS and Android platforms, either through hybrid frameworks like React Native or native app development.
- Continuous integration and deployment: Implementing automated testing, continuous integration, and deployment pipelines to ensure a streamlined development process and faster delivery of new features and updates.
- Real-time features: Implementing real-time communication features like live updates, chat functionality.

7 GUIDELINES

At the end the guidelines provided by the professor was followed as mentioned below:

1- Clearly state the project's purpose and why the MERN stack was chosen.
   - ✓ This is extensively described in chapter 3 chater 4 .
2- Provide a brief overview of the MERN stack, highlighting the role of each technology.
   - ✓ This is done chapter 1 and described the technologies as well.
3- Outline the development process, emphasizing key methodologies employed.
   - ✓ This is extensively described in chapter 1 and 6
4- Build a web app and briefly explain the primary app features.
   - ✓ The topic solved in chapter 4
5- Suggest future improvements.
   - ✓ This is done in chapter 6.

8 CONCLUSION

This thesis successfully explored the potential of React.js for building complex web applications, focusing on its effectiveness and the challenges encountered during development. The development of Travalluge, a feature-rich travel platform, served as a practical case study to demonstrate these findings.

8.1 Key Findings:

- React.js's component-based architecture and virtual DOM proved instrumental in developing a maintainable, scalable, and performant platform like Travalluge.
- The extensive ecosystem of libraries and tools available for React.js facilitated rapid development and integration of complex functionalities like user authentication, social interaction features, and content management.
- While React.js offers significant advantages, the learning curve and potential state management complexity require careful consideration for large-scale applications.

8.2 Travalluge: A Testament to React.js Effectiveness:

Travalluge exemplifies the effectiveness of React.js in building a user-friendly and dynamic web application. The platform empowers travelers by offering:

- Features with a Unique Approach like Upvote and Downvote
- Interactive travel planning tools
- Engaging content creation functionalities
- Social interaction features for a thriving community

The MERN stack provided a robust foundation for Travalluge, ensuring scalability and efficient data handling.

8.3 Looking Forward:

This thesis not only contributes to the understanding of React.js for complex applications but also presents Travalluge as a valuable platform for the travel community. Future enhancements could include a mobile app for on-the-go travel planning and integration with travel booking services for a seamless user experience.

In conclusion, this thesis successfully demonstrated the power of React.js in crafting a feature-rich travel platform. It provided valuable insights into the effectiveness of React.js for complex web development and paves the way for further exploration of its potential in building dynamic and engaging user experiences.

BIBLIOGRAPHY

[1] JWT: JSON Web Token [online] [cit. 24.04.2023]. Available from : https://jwt.io/introduction

[2] IBM: what's MongoDB?[online] Available from : https://www.ibm.com/topics/mongodb

[3] MongoDB : [online]Available from : https://www.mongodb.com/database-management-system

[4] Digital Varys: Introduction to mongo DB diagram[online]Available from: https://digitalvarys.com/introduction-to-mongodb/

[5] GoogleTrends [online]. [cit.04.04.2024]. Available from: https://bit.ly/ReactGraph

[6] HTML.com: [online] Available from: https://html.com/html5/

[7] Daniel Puiatti : Mastering HTML [online] [cit.25.04.2023] Available from : https://danielpuiatti.com/mastering-html-the-essential-concepts-for-aspiring-developers/

[8] Gouravsasson.Hashnode: Introduction to Web and Html [online] [cit.25.04.2023] Available from : https://gouravsasson.hashnode.dev/introduction-to-web-and-html

[9] COPES, Flavio. The HTML Handbook [online]. [cit. 25/04/2023]. Available from: https://flaviocopes.com/book/html/#22-tags-vs-elements

[10] MDN. The HTML [online]. [cit. 25.04.2023]. Available from: https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/The_head_metadata_in_HTML

[11] Computer-PDF. CSS [online]. [cit. 25.04.2023]. Available from: https://www.computer-pdf.com/67-learn-css-basics-introduction

[12] COPES, Flavio. The CSS Handbook [online]. [cit. 25/04/2023]. Available from: https://flaviocopes.com/book/css/#1-preface

[13] LaunchSchool: JavaScript [online]. [cit. 25/04/2023]. Available from: https://launchschool.com/books/javascript/read/introduction

[14] Dave Thau JavaScript: The Book of JavaScript , San Francisco ,2000

[ISBN: 1886411360].

[15] Flanagan D. JavaScript: The Definitive Guide, Fifth Edition O'Reilly, London, 2006.

[ISBN 10: 0-596-10199-6].

[16] Joe Morgan React.js: How To Code in React.js New York, USA 2021.

[ISBN 978-1-7358317-4-9].

[17] Cory Gackenheimer React.js: Introduction to React New York, USA 2015.

[ISBN-13 (electronic): 978-1-4842-1245-5].

[18] React Docs [online] [cit. 25.04.2023]. Available from: https://legacy.reactjs.org/docs/components-and-props.html

[19] React Docs [online] [cit. 26.04.2023]. Available from: https://legacy.reactjs.org/docs/hooks-intro.html

[20] React.dev [online] [cit. 26.04.2023]. Available from: https://react.dev/reference/react/Component#componentdidmount

[21] React.dev [online] [cit. 26.04.2023]. Available from: https://react.dev/reference/react/Component#componentdidupdate

[22] React.dev [online] [cit. 26.04.2023]. Available from : https://react.dev/reference/react/Component#componentwillunmount

[23] React.dev [online] [cit. 26.04.2023]. Available from : https://react.dev/reference/react/PureComponent#migrating-from-a-purecomponent-class-component-to-a-function

[24] React.dev: React JSX [online] [cit. 26.04.2023]. Available from : https://react.dev/learn/writing-markup-with-jsx

[25] LogRocket: React virtual DOM [online] [cit. 26.04.2023]. Available from: https://bit.ly/3OtKUmu

[26] O'reilly: Performing calculations in the Virtual DOM limits rerendering [online] [cit. 26.04.2023]. Available from: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch02.html

[27] javaTpoint: ReactJS Tutorial : State Vs. Props [online] [cit. 26.04.2023]. Available from: https://www.javatpoint.com/react-state-vs-props

[28] React : React Docs: Introducing Hooks [online] [cit. 26.04.2023]. Available from: https://legacy.reactjs.org/docs/hooks-intro.html

[29] W3Schools : React Tutorial: React Router [online] [cit. 26.04.2023]. Available from: https://www.w3schools.com/react/react_router.asp

[30] Redux: Redux Fundamentals : Redux Overview [online] [cit. 01.05.2023]. Available from: https://redux.js.org/tutorials/fundamentals/part-1-overview

[31] Formik docs: Overview: What is Formik? [online] [cit. 01.05.2023]. Available from: https://formik.org/docs/tutorial#overview-what-is-formik

[32] Axios: Getting Started : What is Axios? [online] [cit. 01.05.2023]. Available from: https://axios-http.com/docs/intro

[33] Material UI : Material UI – Overview [online] [cit. 02.05.2023]. Available from: https://mui.com/material-ui/getting-started/overview/

[34] NPM : Yup [online] [cit. 02.05.2023]. Available from: https://www.npmjs.com/package/yup#schema-basics

[35] Nodejs.dev : node.js : Introduction to Node.js [online] [cit. 02.05.2023]. Available from: https://nodejs.dev/en/learn/

[36] Educba : Introduction to ExpressJS [online] [cit. 02.05.2023]. Available from: https://www.educba.com/what-is-expressjs/

[37] MongoDB : Data Model Design : MongoDB [online] [cit. 02.05.2023]. Available from: https://www.mongodb.com/docs/manual/core/data-model-design/

[38] Stefanov, Stoyan. React: Up and Running: Building Web Applications. 2016.

[ISBN 978-1491931820].

[39] Banks, Alex, and Eve Porcello. Learning React: Functional Web Development With React and Redux. 2017.[ISBN 978-1-491-95462-1].

[40] Casciaro, Mario, and Luciano Mammino. Node.js Design Patterns. Packt Publishing Ltd, 2016.[ISBN 978-78588-558-7].

[41] Mardan, Azat. :Express.js Guide 2014.[ISBN 9781494269272].

[42] Kristina Chodorow and Michael Dirolf: MongoDB: The Definitive Guide. “O’Reilly Media, Inc.,” 2010.[ISBN 978-1-449-38156-1].

LIST OF ABBREVIATIONS

HTML	Hypertext markup language.
CSS	Cascading style sheets.
SEO	Search engine optimization.
JSX	JavaScript xml
DOM	Document object model
API	Application programming interface
Xml	Extensible markup language
HTTP	Hypertext transfer protocol
REST	Representational state transfer
HTTPS	Hypertext transfer protocol secured
JWT	JSON web token
JSON	JavaScript object notation
Oauth	Open Authorization
URL	Uniform resource locator
MERN	MongoDB,Express.js,React, Node.js
CDNs	Content delivery networks
PWAs	Progressive Web Apps
ECMA	European Computer Manufacturers Association
OOP	Object-Oriented Programming
UI	User Interface
Props	properties
I/O	Input/Output
NPM	Node Package Manager
SPAs	Single-Page Applications
ORM	object-relational mapping
IoT	Internet of Things
ODM	Object Data Modeling
CRUD	Create, Read, Update, Delete
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
GIF	Graphics Interchange Format
HEIC	High Efficiency Image Format
CORS	Cross-Origin Resource Sharing

LIST OF FIGURES

LIST OF TABLES

APPENDICES

1- My Master Thesis is Web development using React.js - Google Forms
2- Source code“ Travalluge_Te“