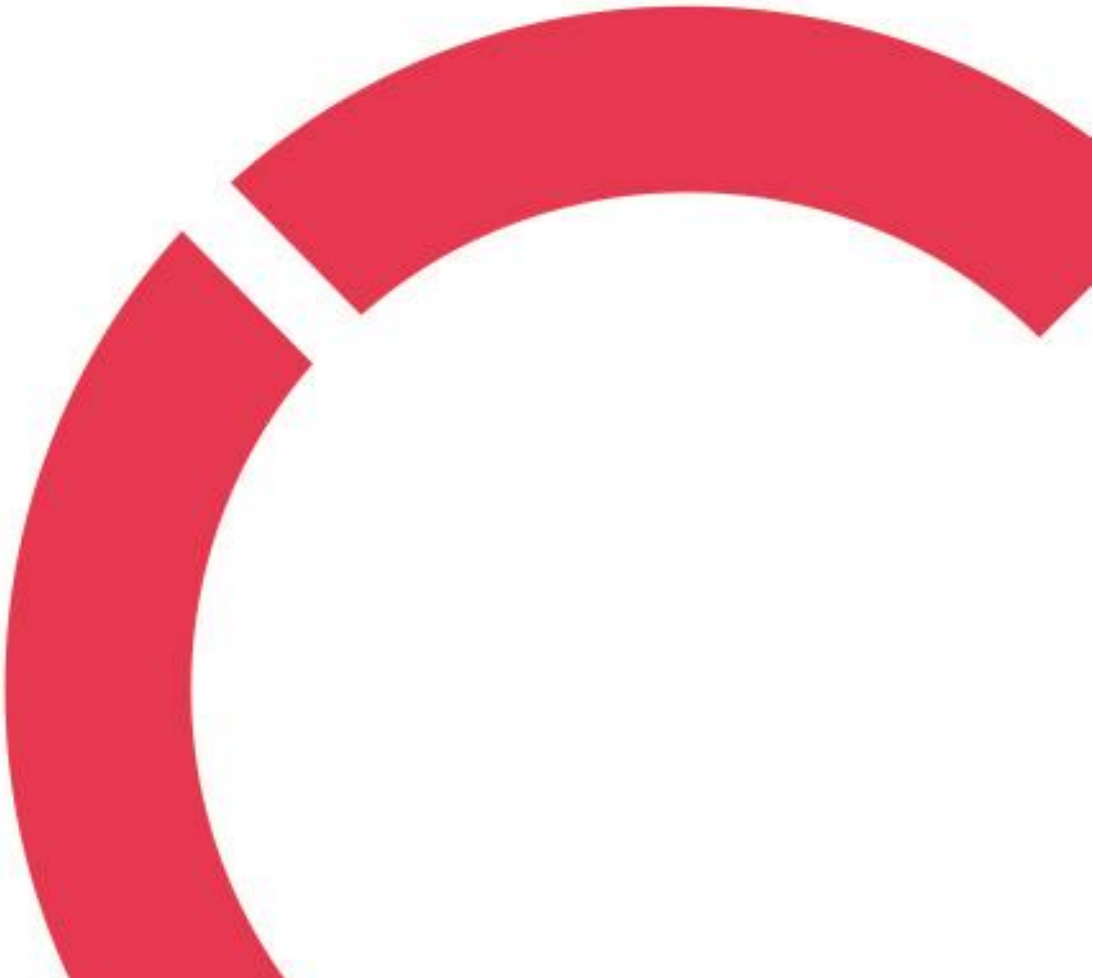
Minh Nguyen

FULL-STACK CRUD APPLICATION

User management system

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Degree Programme
March 2023



ABSTRACT

Centria University of Applied Sciences	**Date** March 2023	**Author** Minh Nguyen
Degree programme Information Technology		
Name of thesis FULL-STACK CRUD APPLICATION		
Centria supervisor Jari Isohanni		**Pages** 38 + 5
Instructor representing commissioning institution or company Jari Isohanni		

The thesis focused on planning and building a user system management web based using Mern stack. The MERN stack which is one of the most widely used in web development and plays a significant role in web development nowadays can eases developers with idea of unifying web development under a single programming language: Javascript. The four components included in MERN stack are MongoDb database, Express.js as back-end web framework, React.js serves as front-end library and Node.js as Javascript environment.

The main purpose of this thesis was to study the usability and functionality of each technology in the MERN stack and as a consequence, to develop a fully functional user system management web application: Create, Update, Delete, and Delete all.

The research conducted during the thesis provides valuable insights into the usability and functionality of each technology in the MERN stack. This knowledge can be applied to future web development projects, allowing developers to make informed decisions about which technologies to use and how to best utilize them. Overall, the development of this user system management web application using the MERN stack showcases the power and potential of this technology stack in modern web development. It provides a clear example of how the four components of the MERN stack can be seamlessly integrated to create a robust and user-friendly web application.

The outcome of this thesis is a fully functional web application that allows users to manage a user system, including creating, updating, deleting, and deleting all users. The application is built using the MERN stack, which allows for a seamless and efficient development process.

Key words
React.js, Node.js, Mongodb, Express.js, HTML, CSS, REST API

CONCEPT DEFINITIONS

List of Abbreviations

API	Application Programming Interface
CDN	Content delivery network
CMS	Content management system
CNCF	Cloud Native Computing Foundation
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
NPM	Node Package Manager
REST	Representational state transfer
UI	User interface
URL	Uniform resource locator

ABSTRACT
CONCEPT DEFINITIONS
CONTENTS

PICTURES

TABLES

1 INTRODUCTION

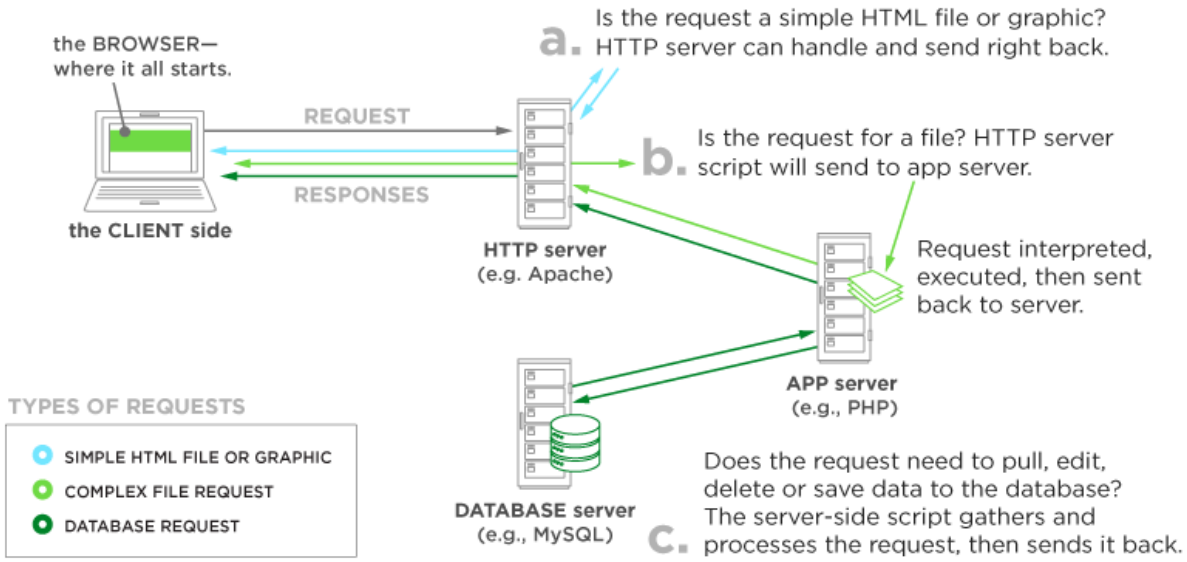
Web development has become an integral part of modern-day businesses and individuals, enabling them to create dynamic and interactive web applications. The MERN stack (MongoDB, Express, React, Node.js) has emerged as a popular technology stack for building efficient and scalable web applications. A simple CRUD (Create, Read, Update, and Delete) application built with the MERN stack can be an excellent starting point for those new to web development. Developing a simple CRUD application with the MERN stack requires a fundamental understanding of each component and how they work together. It involves designing a user-friendly interface, creating a database schema, and implementing CRUD operations. A simple CRUD application can be used to manage a variety of data, such as to-do lists, product inventories, and contact information.

This thesis focuses on developing a simple CRUD application with the MERN stack while addressing the challenges of simplicity and ease of use. The thesis aims to explore best practices and strategies for building a straightforward yet functional CRUD application with the MERN stack. The thesis will essentially be divided into two main parts. The theoretical portion is concerned with providing a thorough understanding of each element of the stack and associated technologies in order to complete the project. The actual use of the management system will be covered later. The end result is a platform with enough capabilities to show how each part of MERN stack connects to one another. User can create, read, update, and delete information in the system. This thesis has provided knowledge that can be applied to create a variety of applications.

2 WHAT IS TECHNICAL STACK AND MERN STACK

Developing a complete application requires more than just writing code, as it also involves creating a database to store data. Once the coding is finished, the next step is to deploy it to a server where it can be compiled and built into a functional application. The finished program comprises not only the code but also the operating system platform, web server, and database. By integrating all these components, an engineering stack can be created to support the application. A structure of stack is usually composed of the following components: Operating system, web server, database and back-end programming language. Each component of the stack performs a different duty.

According to the PICTURE 1, LAMP stack is used to build an application. The flow of the LAMP stack starts with the Linux operating system, which provides the foundation for the other components. Apache serves as the web server, which receives requests from clients and returns responses. MySQL is used as the database management system to store, manage, and retrieve data. Finally, PHP is the server-side scripting language that is used to create dynamic web pages and web applications. When a client makes a request to the web server, Apache receives the request and passes it on to PHP for processing. PHP retrieves data from MySQL and dynamically generates HTML pages, which are then sent back to the client via Apache. The client can then view the requested web page in their web browser (Altamira team, 2020).



PICTURE 1. The working flow in Lamp stack (Altamira team, 2020).

One of the most popular stack recently is MERN stack. It consists of MongoDb, Express.js, React.js, and Node.js as its components. Since MERN integrates four cutting-edge technologies, including Facebook's strong support, it eliminated the need for developers to learn other technologies such as .NET, PHP, Python. The stack is also backed by a large number of open sources packages and a committed community of programmers to boost and maintain software due to the same Javascript platform (Educative, Inc 2022).

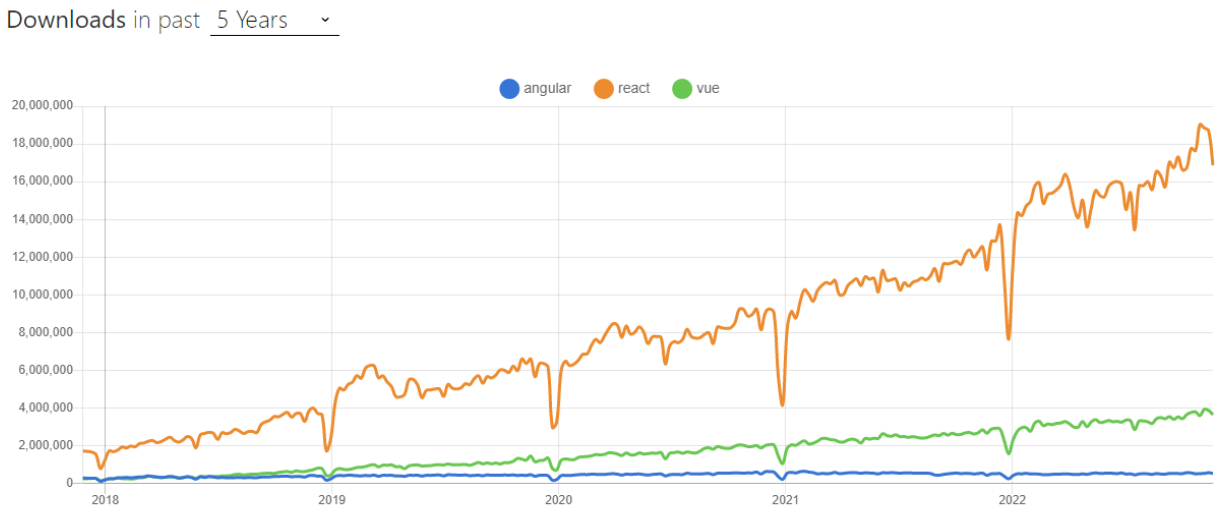
The MERN stack's architecture is shown in PICTURE 2. At the front-end of the program, which is located in the browser, the user initially interacts with the ReactJS UI components. The application backend on a server serves this frontend through Express.js running on top of Node.js. Any interaction that results in a request to alter data is sent to the Express server, which is built on Node.js. Express retrieves data from the Mongodb database as needed and delivers it to the frontend of the application, where it is then shown to the user (Educative, Inc 2022).

PICTURE 2. The 3-tier architecture of Mern stack (Educative, Inc 2022).

2.1 React.js

React is a Javascript front-end library. This open-source library is use to create web user interfaces(UIs). For development firms of sizes, React is a top tool. Declarative, straightforward, and server-side support are some of its key characteristics. It was first introduced in the Facebook newsfeed feature in 2011 by a Facebook software engineer, and Instagram followed a year later. React enables developers to design scalable online apps that can alter data without refreshing the browser, as well as reusable UI components. React launched a number of significant additions that increase its appeal for developers in addition to offering reusable component code, which shortens development time and lowers the possibility of problems and errors: Virtual DOM, JSX and component (Pandit, 2021).

PICTURE 3 compares the quantity of NPM packages downloaded for React with those for Angular and Vue, two other well-liked frameworks. As can be seen, React's popularity has grown significantly over the past 5 years, demonstrating its clear preference.



PICTURE 3. Number of package downloads with NPM in past 5 years of React, Angular and Vue (npm trends, 2022).

2.1.1 Virtual DOM

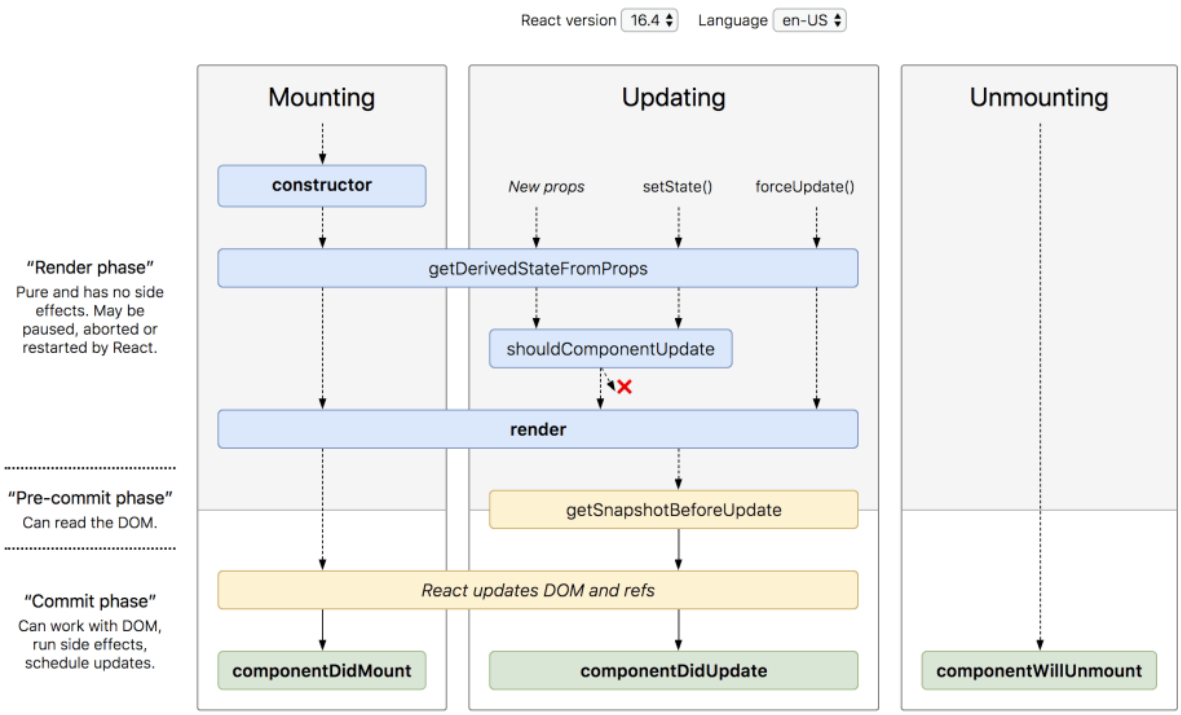
React functions differently than many other front-end JavaScript frameworks in that it creates a virtual version of the DOM (Document Object Model) rather than using the browser's DOM. We refer to the "actual DOM" as a tree of JavaScript objects, representation of an entire HTML document when we use the term "virtual". Every webpage has a DOM representation that is equal to it, which is a tree structure that contains all of the UI elements. Any tiny change to a state in the UI necessitates updating the related DOM representation and re-rendering the UI. The only expensive part of updating the DOM is rendering and re-rendering the user interface. Given the size of the majority of SPAs (Single Page Applications), displaying the UI takes a long time not because the components are sophisticated but because the tree structure that contains them is large (Mojeed, 2022).

React library always maintains twos copy of Virtual DOM; the first and the second. React will keep note of modifications whenever a state in the user interface changes and begin batching them. These modifications are made to the second iteration of Virtual DOM rather than the first DOM. React will

run a diff operation between the first and second versions at a certain time, after which a delta will be calculated. Later, this delta will be applied to the actual DOM so that just the necessary portions of the web page are updated rather than having to be completely repainted (Mojeed, 2022).

2.1.2 Data exchanges and Lifecycles

React components have a unique lifecycle that can be divided into three phases: mounting, updating, and unmounting. During the mounting phase, the component is created and added to the DOM. At this stage, the constructor method is called, followed by the render method, which generates the comp nent's HTML representation. Finally, the component is added to the DOM. In the updating phase, the component is re-rendered when its state or props change. The lifecycle methods such as shouldComponentUpdate, componentDidUpdate, and componentWillReceiveProps can be used to update the component as needed. Finally, during the unmounting phase, the component is removed from the DOM. The componentWillUnmount method can be used to perform any necessary clean-up operations before the component is destroyed (reactjs.org 2019). The React.js lifecycle functions and when they are triggered are shown in PICTURE 4.

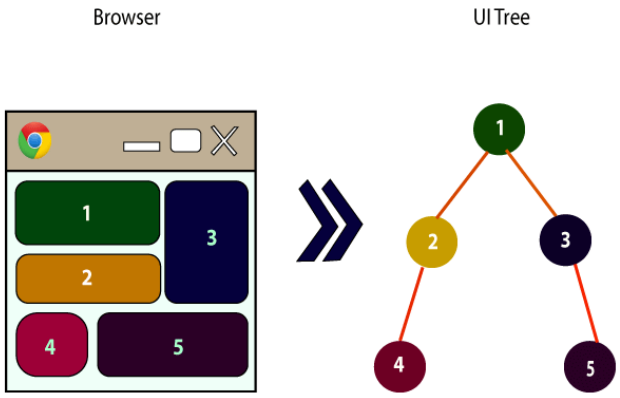


PICTURE 4. Lifecycle methods in React (reactjs.org 2019).

2.1.3 Components

The component-based methodology used in React is designed to make it easier to manage and update large-scale single-page applications. In traditional DOM structure, changes made to one part of the application can have unintended effects on other parts of the application, making updates and maintenance difficult and time-consuming. By breaking down the application into smaller, reusable components, developers can isolate specific parts of the application and make changes more easily. Each component has its own code, state, and rendering logic, making it easier to debug and maintain. The component-based methodology also encourages a more modular approach to development, with each component having a well-defined interface and clear responsibilities. This makes it easier to understand and maintain the application as a whole, even as it grows in complexity over time (Tutorialspoint, 2023)

In React, a component can be thought of as a modular and reusable block of code that defines a specific part of a user interface. Components can be combined to create more complex user interfaces, making it easier to manage and organize code. A parent component can unify and manage all of its child components, even if they all share the same area. This allows for greater flexibility and control over the layout of the user interface. Additionally, by breaking down the user interface into smaller, more manageable components, developers can make their code more maintainable and easier to debug. React components are also designed to be reusable, meaning that they can be used in multiple parts of an application. This can save time and effort when building large and complex applications, as developers can simply reuse existing components instead of creating new ones from scratch. PICTURE 5 illustrates the structure of React components. The root is the initial component. Each of the other parts develops into a branch, which is then further subdivided into sub-branches (javapoint, 2022).



PICTURE 5. UI structure in React.js (javapoint, 2022).

2.1.4 Functional Component

In essence, functional components are defined in the form of a JavaScript function that may take certain props (properties) as an argument and return a React element. Functional components are entirely presentational. Functional components always provide the same outcome when given the same props because of their predictability and simplicity. Therefore, wherever possible, developers choose functioning components. All functional components are referred to as stateless components or dumb components since they are naturally incapable of managing any state and are only responsible for producing user interface elements. PICTURE 6 is an example of a functional component (Hamedani, 2018).

```
function Hello(props){
   return <div>Hello {props.name}</div>
}
```

PICTURE 6. Functional component (Hamedani, 2018).

2.1.5 Class Component

Components in React may also be specified as classes. Class components are produced using the ES6 class syntax, as their name would imply, and they differ from functional components in several ways.

Component classes must first be extended from a component class, which provides access to all the parent component's functionalities. In order to return a React element, they need a render() function (JSX code). In addition to the render() function, class components can also have Lifecycle methods. Due to the constructor function that they take, where they may accept props and retain their own data with state, class components are also known as stateful components. Last but not least, class components have instances, so you may access props with “this.props”. The render method must be defined in every "React.Component" subclass, as seen in PICTURE 7. Due to the starting state and the fact that super(props) should be called before any other lines inside the constructor, a constructor function is built here. Incorrect prop definitions might result in problems if this is not done (Vishnupriya, 2022).

PICTURE 7. Class component.

2.2 Node.js

Node.js is an open source, cross-platform runtime environment and library that is used for running web applications outside client’s browser. Although it was first created with real-time, push-based architectures in mind, it is utilized for server-side programming and is generally employed for non-blocking, event-driven servers, such as conventional web pages and back-end API services. Every browser has a unique JavaScript engine version, and node.js is based on the V8 JavaScript engine built in Google Chrome. Node.js is unique in that it is a runtime environment rather than a framework or library, as is the case with typical application software. A developer can use Web APIs in a runtime environment, sometimes known as RTE, to create code. A JavaScript engine is then used to read the code. Because

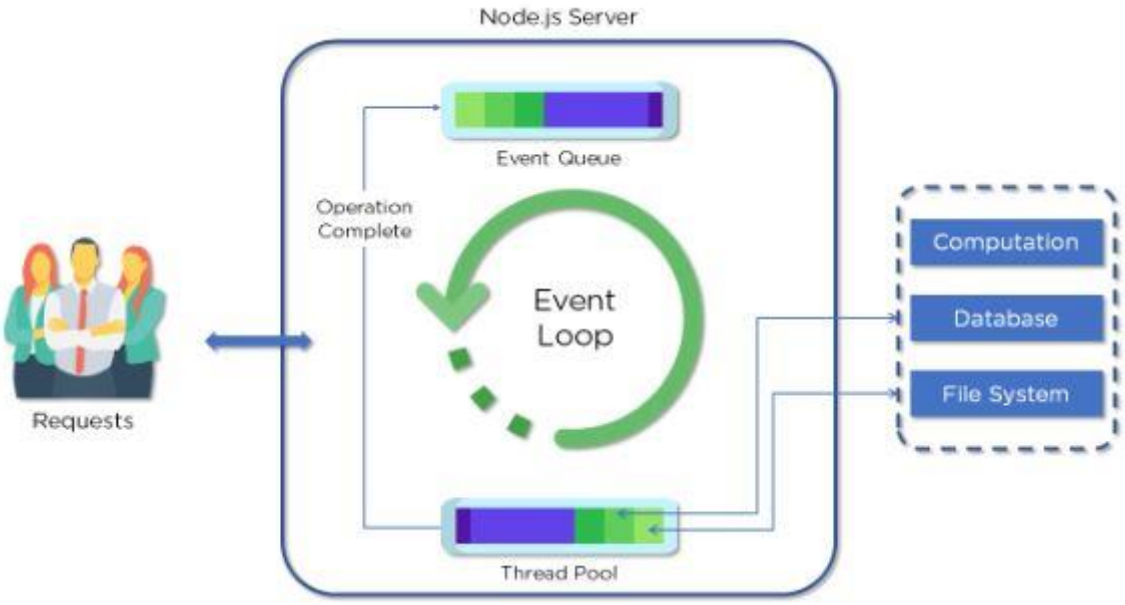
of its small weight, adaptability, and ease of deployment, the application project may be optimized and accelerated (Heller, 2022).

2.2.1 Node.js architecture

The "Multi-Threaded Request-Response" architecture is often used in Web applications created without Node JS. Request/Response Model is the short name for this model. The server receives a request from the client, processes it in accordance with the request, then prepares and delivers the client a response. HTTP protocol is used by this model. This Request/Response model is a Stateless Model, same as HTTP, which is a Stateless Protocol. However, this approach manages several client requests at once by using Multiple Threads (Sufiyan, 2022).

When using multiple-thread processing, one is chosen for each request until all of the available threads have been utilised. The server must then wait for a busy thread to become free once more when this occurs. Applications may become sluggish and ineffective as a result, which may have an impact on anything from lead conversion rates to customer satisfaction. It may become an issue more so if the program must handle a large volume of concurrent client requests (Sufiyan, 2022).

The Request/Response Multi-Threaded Stateless Model is not used by Node.js Platform. It follows Single-Threaded with Event Loop Model as PICTURE 8 described. The fundamental distinction between the two is that in single-thread systems, each request is handled by a single main thread, and blocking input/output activities are carried out in a non-blocking manner by employing event loops. A single-thread architecture can, in theory, perform and scale much more quickly and efficiently than multiple-thread setups (Taha Sufiyan, 2022).



PICTURE 8. Node.js architecture (Taha Sufiyan, 2022).

2.2.2 Node Module

Node.js has a variety of 'modules' that are maintained within specific contexts to prevent them from interfering with one another or contaminating the overall scope of node.js. Open-source software depends on this. A capability in Node.js that is structured into JavaScript files and reusable across the Node.js application is referred to as a module. Within Node.js, modules can be classified as Core Modules, Local Modules, or Third-Party Modules. The foundational, essential features of Node.js are included in Core Modules. They are a component of the Node.js binary distribution and load automatically when a node process launches. The Node.js application can construct its own modules, known as local modules. In addition to the basic feature package, they also comprise alternative and extra functions in separate files and directories. Additionally, local modules may be packaged and made available to the larger Node.js community for usage. A third-party module is pre-written code that can be loaded into your Node.js application to increase its functionality and add new features (Bennison J, 2022).

2.2.3 Node Package Manager

NPM, or Node Package Manager, stands for. It is entirely built in Javascript and is the Node.js default package manager. All of the dependencies and modules are managed through NPM, a command-line

client for Node.js. It is included in the system when Node.js has been installed. NPM is used to install the necessary packages and modules in node projects. Modules are JavaScript libraries that may be added to a Node project based on the project's requirements. A package contains all of the files needed for a module. It comes with a lot of libraries that are great resources for Node.js developers and speed up the creation of new applications. The package.json file may be used by NPM to install all of a project's dependencies. It is capable of both upgrading and removing packages. Developers may automatically update their packages while preventing undesirable breaking changes by utilizing the semantic versioning technique in the package.json file to provide a range of valid versions for each dependency (Sonia M, 2021).

2.2.4 Event-driven programming

Applications using event-driven programming can react to different user involvement patterns. A "programming paradigm" is one in which events control the direction in which a program is executed. Events in this sense refer to any user activity, including keystrokes or mouse clicks. They might also refer to messages from different applications or threads. The goal of event-driven programming is to process activities by building event-handling routines and detecting them as they happen. This makes users' experiences more responsive and straightforward, and it gives programs more flexibility when processing several requests at once (Kolesnikova T, 2022).

2.2.5 Event Loop

Despite JavaScript's single-threaded nature, the event loop enables Node.js to conduct non-blocking I/O operations by offloading tasks to the system kernel wherever possible. The majority of contemporary kernels support several background operations since they are multi-threaded. The event loop, in general, is a mechanism in a program that watches for and sends out events or messages. Event loops are the primary control flow components of Node.js. For instance, if a request is due to be handled, it is placed on the event loop and is handled as soon as it is prepared. Node, instead of doing it on its own, delegates the responsibility of handling the system. Because of such behaviour, Node is not actively waiting for this task to finish and can handle other requests in the meantime. The event loop makes Node.js faster and more efficient than other technologies (Zaig R, 2019).

2.3 Express.js

Node.js released the open-source, free online application framework known as Express.js. The Express.js framework is designed to design apps rapidly and efficiently without skipping any crucial aspects. The majority of web apps may be found online and are accessed by using a web browser. Because Express.js is based on JavaScript, it is extremely simple for programmers and developers to create APIs and online apps with it. Since Express.js is a member of the Node.js family, most of the code and documentation for it have already been produced. The code allows programmers to create multi-page, single-page, or hybrid web applications. Express.js is a compact framework that makes it possible to better structure web programs on the server-side (Flexible, 2023).

2.3.1 Routing

Routing is a fundamental aspect of building web applications, as it defines how the application responds to incoming client requests. In addition to the URI and HTTP request method, routing may also consider additional factors such as request headers, query parameters, and request bodies. When a client makes a request to a specific endpoint, the application's routing system determines which handler function should be called to handle the request. This handler function may perform a range of tasks, such as querying a database, processing data, or returning a response to the client. Routing can also be used to define middleware functions, which are functions that are executed before the main handler function is called. Middleware functions can perform tasks such as authentication, logging, or data validation, and can modify the request or response as necessary (Chaddha, 2022).

PICTURE 9 describe how a route works in application to defined for GET request. App is an instance of Express. “app.get” is the HTTP request method. “/home” is the route and finally (req, res) => {} is the handler function, which runs in response to a match request.

```
const express = require('express');
const app = express();

// GET method route
app.get('/home', (req, res) => {
   res.send('OK');
});
```

PICTURE 9. An example of a route that is defined for GET request.

2.3.2 Middleware

Express.js is a middleware and routing framework that handles various web page routing and operates in between the request and response cycle. Before the controller operations transmit the answer, middleware is done once the server receives the request. Before the server sends a response, middleware has access to the request object and replies object and may process the request. Each middleware function call in an Express-based application is a separate entity (Selvaganesh, 2018).

The basic syntax for the middleware functions is shown in the PICTURE 10. Three parameters are required for middleware functions: the request object, the response object, and the subsequent function in the request-response cycle of the application, or two objects and one function. Middleware functions often add information to the request or response objects and perform some code that may have unintended consequences for the application. They can also break the loop by responding when a certain need is met. If they don't send a response when they're finished, they begin the stack's subsequent function's execution. This causes the third argument to be called next() (Selvaganesh, 2018).

```
app.get(path, (req, res, next) => {}, (req, res) => {})
```

PICTURE 10. Middleware syntax.

2.4 MongoDb

MongoDB is an open-source NoSQL database management program. NoSQL databases are used in place of conventional relational databases. Working with sizable, dispersed data sets makes good use of NoSQL databases. MongoDB is a technology that can manage, store, and retrieve information that is document oriented. Various types of data are supported by MongoDB. It is one of the various NoSQL nonrelational database technologies that emerged in the middle of the 2000s. Typically, big data applications and other processing tasks requiring data that doesn't fit well in a rigid relational model employ this technology. The MongoDB design uses collections and documents as opposed to relational databases' use of tables and rows. (Vaughan, 2022).

2.4.1 Querying

The query language for MongoDB is based on JSON, which stands for Java Script Object Node, while the database itself is based on JavaScript (JavaScript Object Notation). The querying process is made noticeably simpler when JSON is used as the query language. A document may be created, deleted, retrieved, updated, or searched for by defining the kind of action in a JSON object, which also enables the developer to build more automated searches. On the other hand, while the query language is "English like" and a specific record may be obtained using query phrases like SELECT or WHERE, the SQL database is more constrained when it comes to programmed queries. When data is stored as an object rather than relations or tables, as they are in a more conventional SQL database, the developer has more freedom to structure models (Vaughan, 2022).

2.4.2 Mongoose

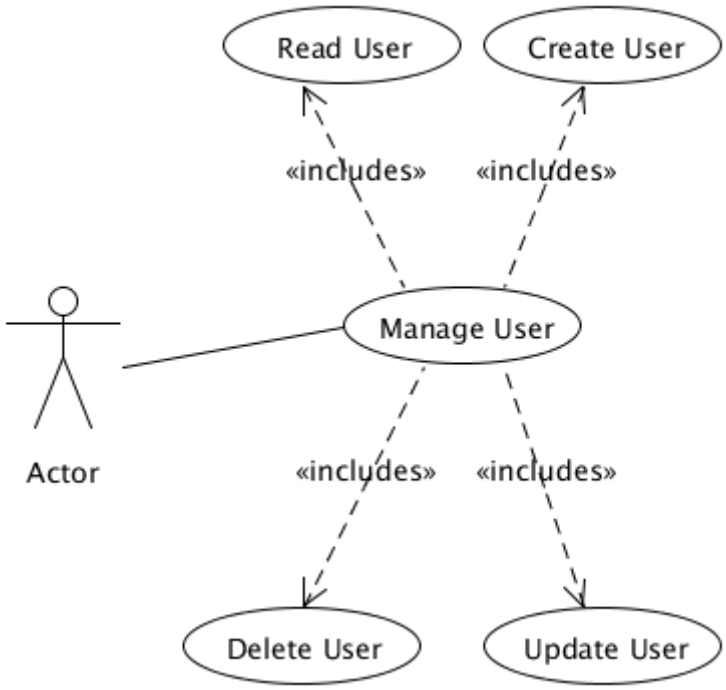
Mongoose is a query language that facilitates data reading, writing, updating, and deleting activities between the server and database. It is an Object Data Modelling (ODM) framework for MongoDB and NodeJS that offers schemas to represent application data, clearing out the ambiguity of databases in the process. By employing schema, it applies a uniform structure to all the documents in a collection. Additionally, it verifies the information that is entered in the papers and permits only true information to be kept in the database. Overall, Mongoose offers all of MongoDB's functionality, including tools for query creation and business logic in the data (Bormon, 2022).

2.4.3 Scalability

Developers have two alternatives when it comes to data volumes and the scalability of an application: either scale up, which requires them to invest in a larger computer capable of holding more data, or scale out, which is a word for distributing the data over several machines. When scaling up reaches the physical limits of the hardware, it might become too expensive to buy a large new system. MongoDB is made to scale out; it divides the data among several servers and evenly distributes the data load. The data is read from and written to the relevant computer using Mongo's routing scheme. Due to the flexibility of this architecture, more servers and storage capacity may be added as needed.

3 PROJECT IMPLEMENTATION

User management system is a simple web application project designed using the MERN stack which facilities the user to create, search user by name, update, delete the list of the user's account. The project also replicates the process of building a website, including crucial steps like building the front-end, the back-end, the database, and connecting the various components. The front-end of the user management system is built using React, which provides a fast and responsive user interface that can be easily customized and extended. The back-end of the system is built using Node.js and Express, which provide a fast and efficient way to build RESTful APIs for handling user data. The database for the user management system is built using MongoDB, which is a flexible and scalable NoSQL database that can handle large amounts of data. The use case diagram of the application is shown in the PICTURE 11 below.



PICTURE 11. Usecase diagram of User management system.

3.1 Environment Setup

To complete any activity, it is always crucial to select the appropriate equipment and setting. When these tools and surroundings are used together, work is more productive, efficient, and time-saving. One of the keys to success in the development process and making it simpler is picking a comfortable stack. The software was created entirely on a Windows laptop. In addition, a number of third parties' tools and software were employed to speed up the development process and achieve the desired outcome.

For this project, the coding editor of choice was Visual Studio Code. A Microsoft-led open-source initiative that was originally made available in 2015 was called VS Code. Since then, among developers, it has grown to be one of the most widely used code editors. VS Code is a fantastic toolkit that includes setup, UI for displaying file content, customisable settings, debugging a NodeJS web program, and cloud deployment of the web app (Code, 2020).

3.1.1 Version control

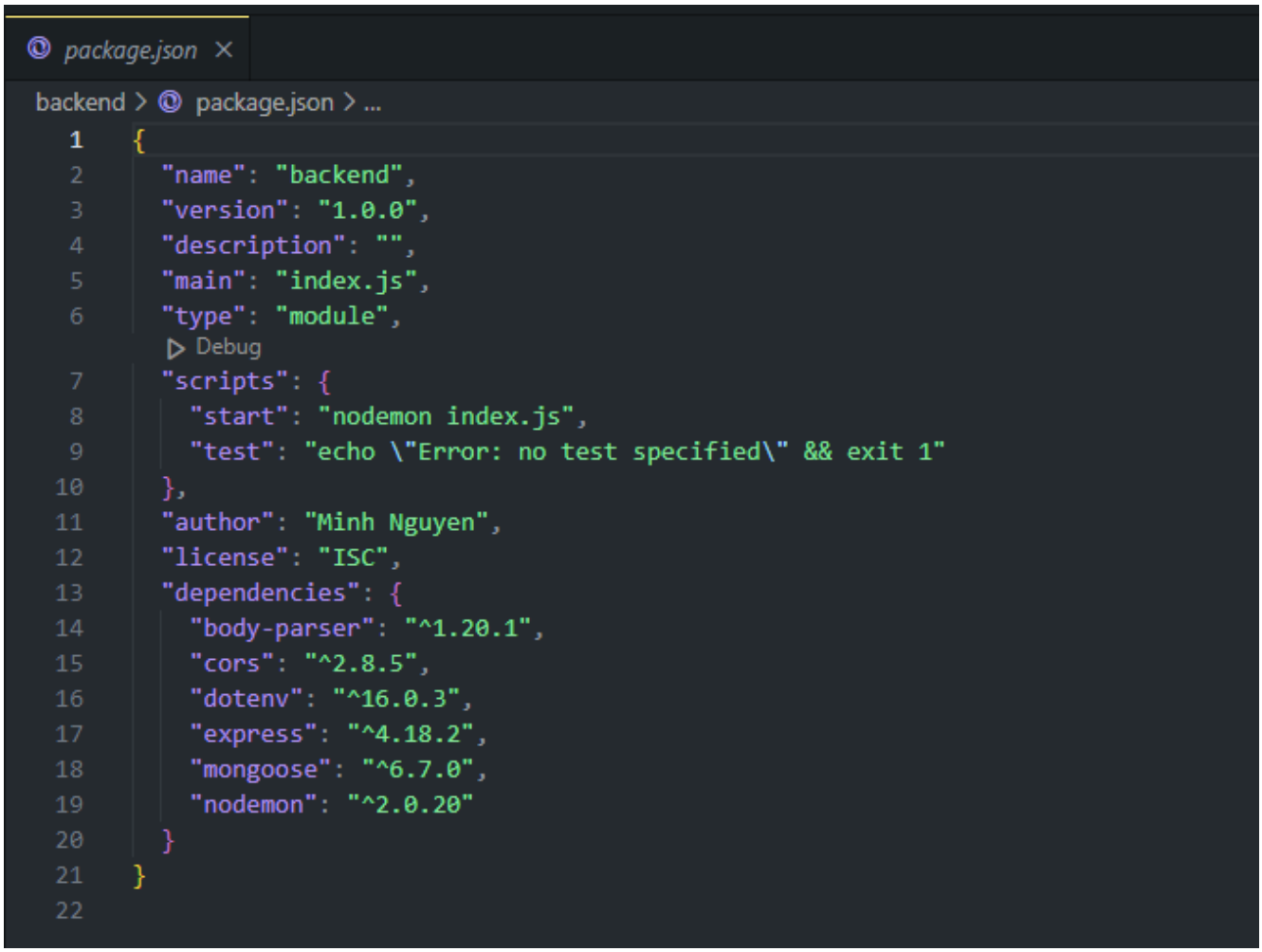
Software configuration management includes version control, which keeps track of changes made to a document or group of documents over time so that the particular versions may be remembered later. Changes are denoted by a code, either a number or a letter, known as the "revision number," "revision level," or simply "revision." For instance, "revision 1" refers to the original collection of files. The set that results from the alteration is known as "revision 2". Each version has a timestamp and the altering party listed next to it. Revisions may be compared, restored, and, in certain cases, combined with other types of files. Collaboration among a team or group working on a project is facilitated by version control. Git, SubVersion, Team Foundation Version Control, and other version control programs are only a few of the numerous options available. Git, a distributed version control system that places an emphasis on speed, data integrity, and support for distributed non-linear workflows, is one of the most popular version control systems and was chosen for the project (Blischak J, Davenport E, Wilson G, 2016).

The benefit of Git is that, by virtue of its architecture, it offers backups. Every Git repository contains the complete repository, unlike other VCSs that have a "single source of truth." Developers typically have two repositories: one hosted in a Git cloud service like GitHub, GitLab, or BitBucket, and one

repository that is a working copy that is kept locally on their machines (Blischak J, Davenport E, Wilson G, 2016).

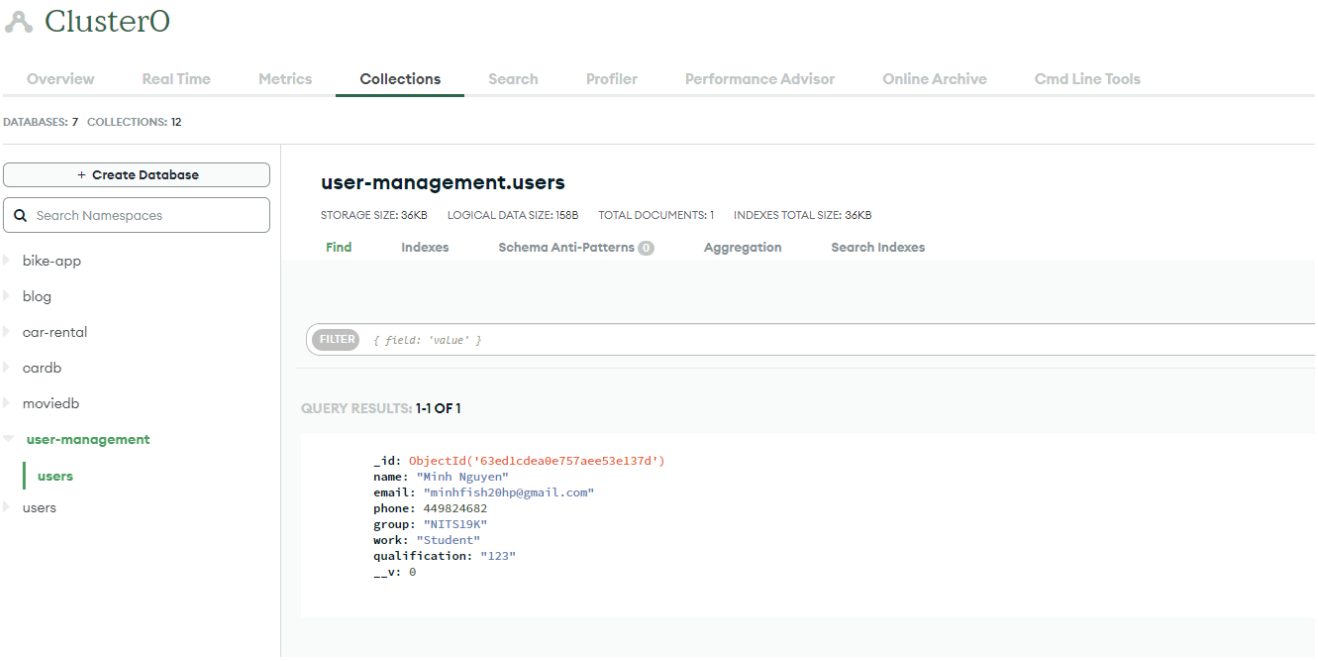
3.1.2 Framework and Node Packages installation

In the window context, NodeJS version 18.12.0 was installed from the official website. The "npm init" command was used to launch the program. During implementation, the application's name, version, and project description were provided. The data supplied during the download was immediately used to create a file with the designation "package. json". The command "npm install express" loaded the Express framework version 4.18.2 after installing NodeJS. The next stage is to use mongoose to connect the server and database and establish communication. The most recent version of the mongoose, which was 6.7.0 at the time of installation, was obtained by using the command "npm install mongoose." The URL, login, and password established during the MongoDB Atlas setup were used to establish a database link between the program and the server. Other packages such as “cors”, “nodemon” also needed to be installed in this project. The structure of “package.json” file is shown in the PICTURE 12 below.



PICTURE 12. The structure of “package.json” file.

Installing MongoDB locally on the computer the application is being created on or straight onto the MongoDB Atlas as a cloud solution are both options. On the MongoDB website, a new account was made. Creating a new project resulted in the creation of a complimentary MongoDB server. In addition, the database was given read and write permission, which was used by the NodeJS application to receive and write data to and from the database. Furthermore, in order for the app to interact with the server, the machine's present IP address was added to the IP whitelist. The query "npm install mongodb" was then used to complete the installation of the MongoDB driver in the program. The overview of a cluster for the application is shown below in PICTURE 13.

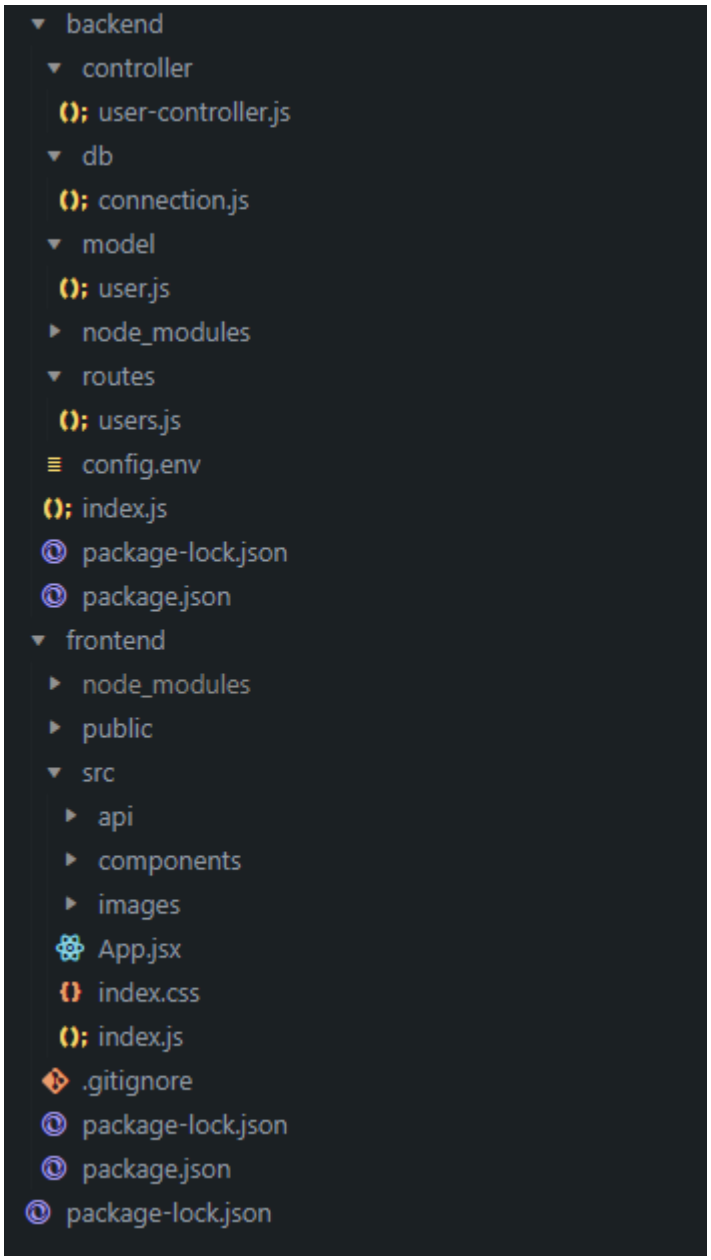


PICTURE 13. The mongoDb Cluster of the application.

Finally, React 17.0.2 was the version loaded during the project's development. React was used to create the application's user-friendly UI because it symbolizes the separation of client-side code. Additionally, React was used to reuse all of the project's components and lessen the tension associated with creating CSS formatting from inception.

3.1.3 Code structure of the application

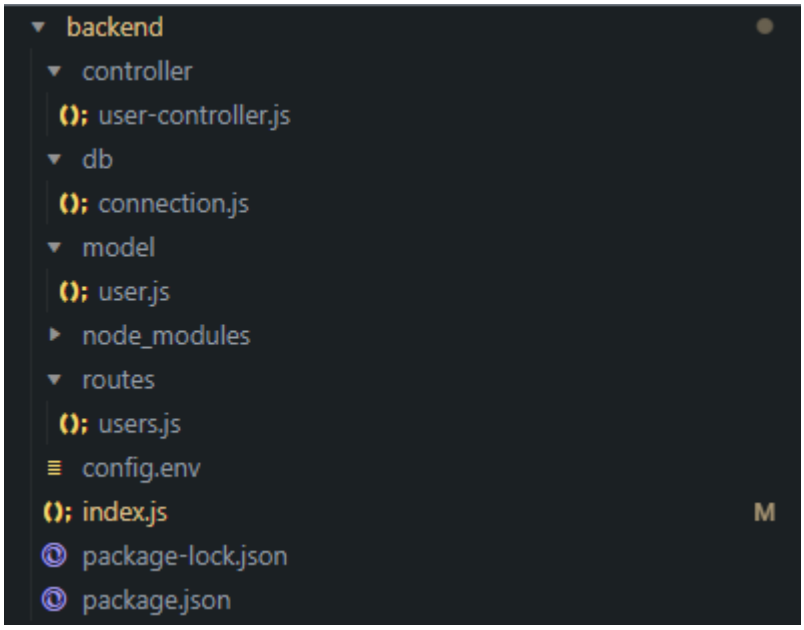
All the tools and packages were initially loaded during the development process. The actual coding of the program began after early setup. The application's end file and folder layout is shown in the PICTURE 14 below.



PICTURE 14. File and folder structure of the application.

There are two files, "frontend" and "backend," in the application's folder layout mentioned above. The files and folders for the front end are all contained in the frontend folder, and the files and folders for the back end are all contained in the backend folder. The usual, automatically generated "node modules" for ReactJS contain the primary source code under the "src" folder. The "App.js" file, where set-up takes place, is the project's starting point. Later, there will be a more detailed discussion of each folder's function.

3.2 Back-end Implementation

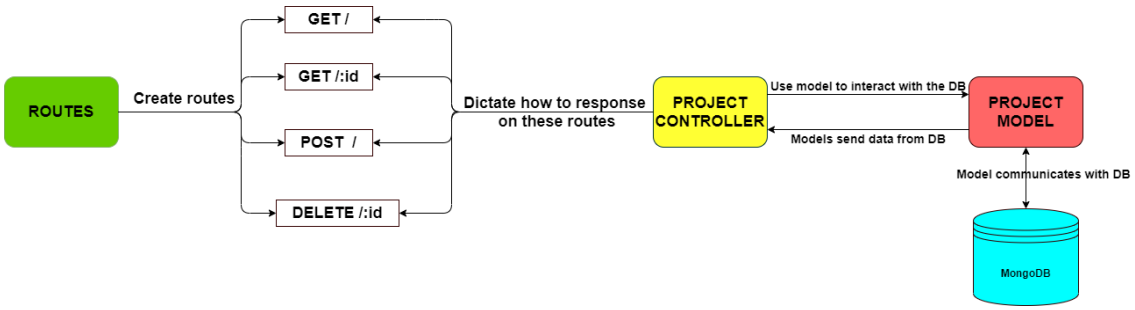


PICTURE 15. Backend folder.

The PICTURE 15 shows the folder structure of the back-end. The controller folder is used to create functions for the application. The db folder is used for connecting to the database. The routes folder is used for manages endpoints for the application. The "config.env" file is used to store sensitive environment variables such as passwords, port, API credentials and additional details that shouldn't be explicitly written in code. The "index.js" file is a convention in Node.js for defining the main entry point for a module or application. The content of "index.js" file is shown in the PICTURE 16 below. The relationship between these folder can be presented in PICTURE 17.

```
backend > (); index.js > ...
1  import express from "express";
2  import dotenv from "dotenv";
3  import cors from "cors";
4  import userRoutes from "./routes/users.js";
5  import bodyParser from "body-parser";
6  import connectDB from "./db/connection.js";
7
8  dotenv.config({ path: '../backend/config.env' })
9
10 const app = express();
11
12 connectDB();
13
14 app.use(bodyParser.json({ extended: true }));
15 app.use(bodyParser.urlencoded({ extended: true }));
16 app.use(cors());
17
18 app.use("/users", userRoutes);
19
20 const PORT = process.env.PORT;
21
22 app.listen(PORT, () => {
23   console.log(`Server running on PORT ${PORT}`);
24 });
25
```

PICTURE 16. The “index.js” file.

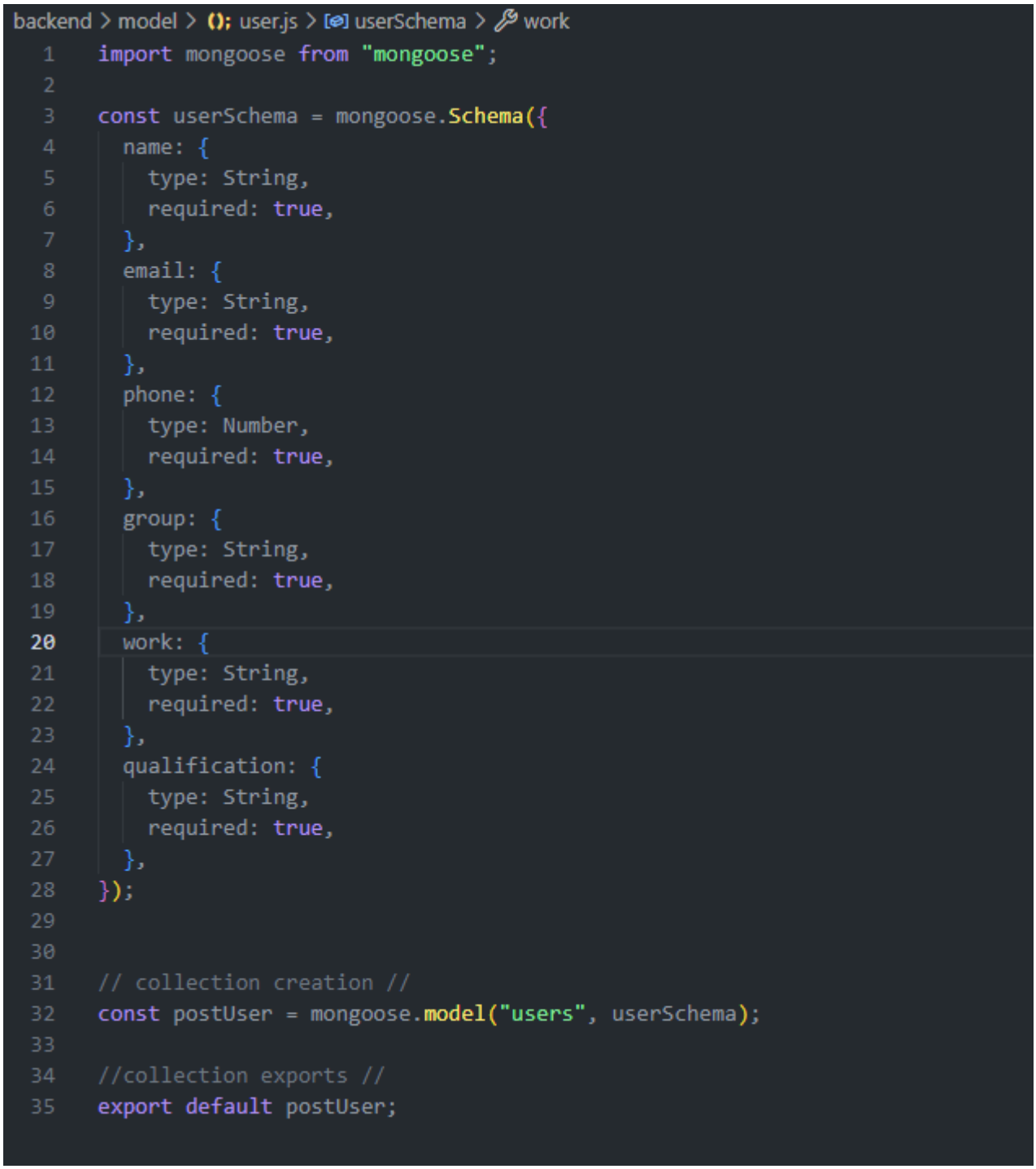


PICTURE 17. Relationships.

3.2.1 User’s model

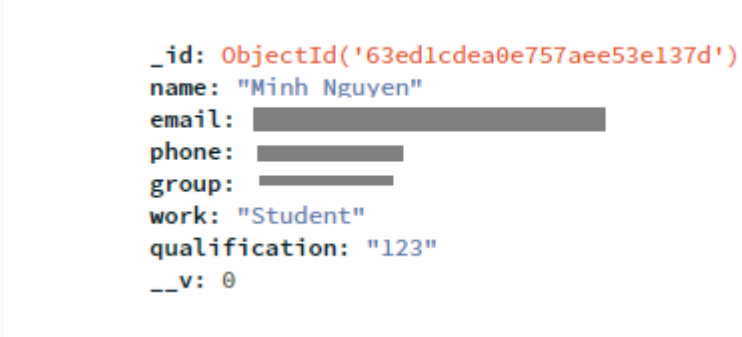
Defining models is an essential step in developing a MongoDB-based application. It helps developers to define the schema of the data, as well as the relationships between the different data types in the

application. In the project, the User model was defined to establish a consistent structure for storing user information. The User model defines the properties of a user document, such as their name, email, and group. These properties are used to create new user documents, as well as to retrieve and update existing user information. By defining a standard structure for user data, the User model ensures that all user information is consistent and can be easily accessed and manipulated by other program components. The User model is presented in the PICTURE 18 below.



PICTURE 18. User's model.

The User model is defined in JSON format, which is a lightweight data interchange format that is easy to read and write. JSON allows developers to define complex data structures using a simple and concise syntax, making it an ideal choice for defining models in MongoDB-based applications. The JSON format for User is presented in the PICTURE 19.



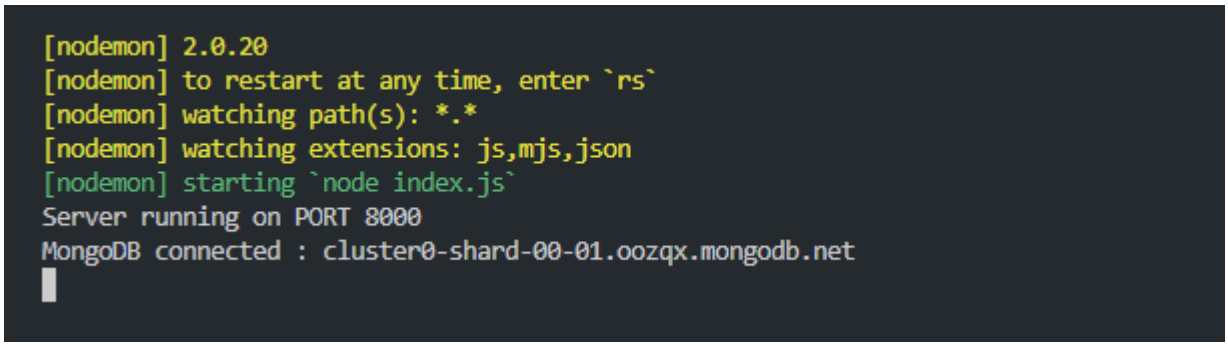
PICTURE 19. JSON format for User.

3.2.2 Database

Since MongoDB is where the data is stored, a link between the server and database had to be established. This section uses Mongoose because it has the tools needed to interact with MongoDB. Although MongoDB can be accessed using the Mongo driver, Mongoose offers greater convenience. The database connection is shown in PICTURE 20.

PICTURE 20. The database connection.

The names of the techniques are self-explanatory. If the database connection is effective, the server logs success messages; otherwise, it shows error messages. PICTURE 21 shows the connection is successful.



PICTURE 21. Successful connection.

3.2.3 Controller and Routes

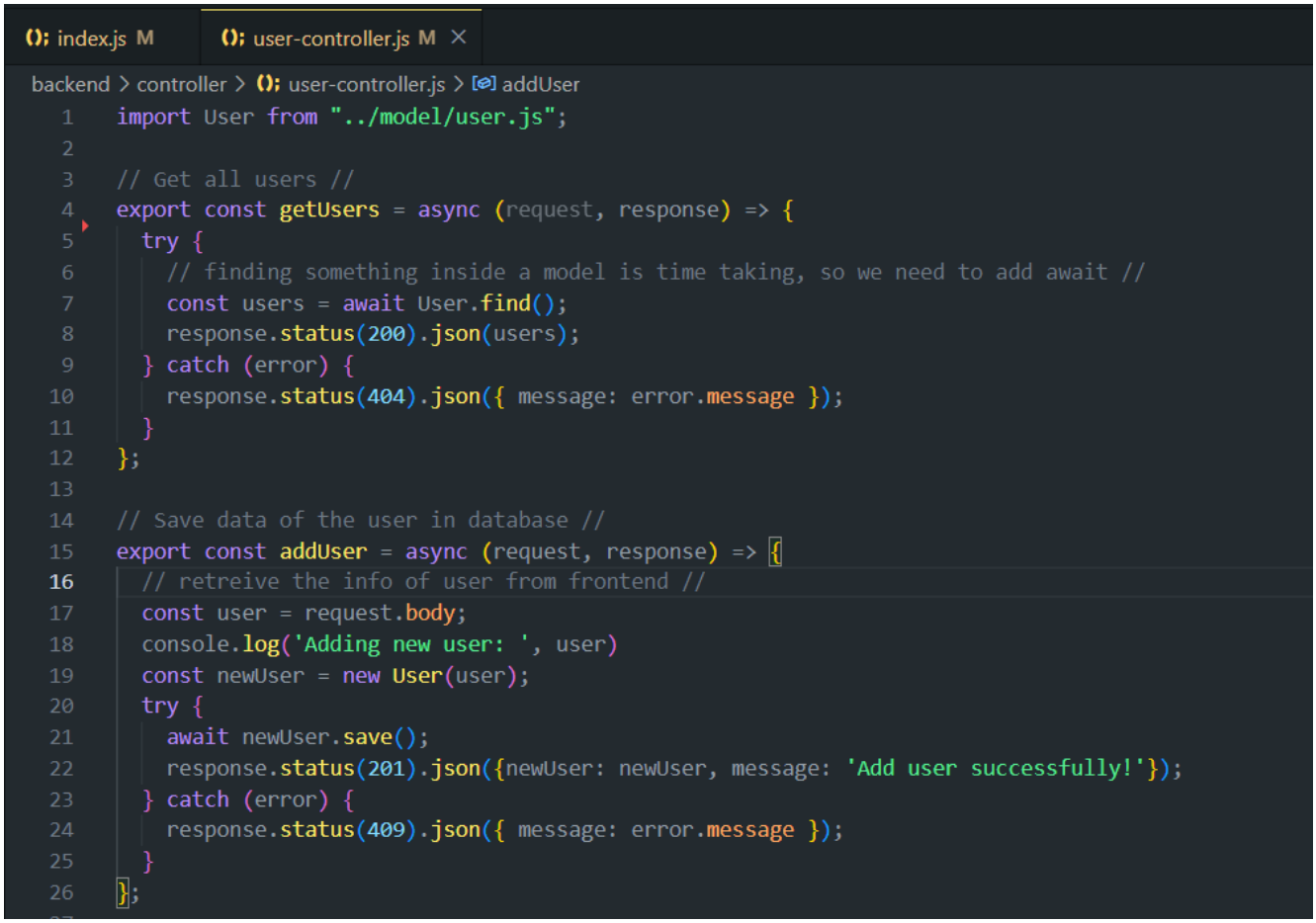
In the routes folder, router object is exported with several routes to handle HTTP requests related to user management application. The router has six routes, each of the six routes calls a separate function. All of the function are exported from controllers folder: “getUsers”, “addUser”, “getUserById”, “editUser”, and “deleteAllUser”.

TABLE 1. List of all routes for user management.

Method	**Route**	**Purpose**
GET	/api/	Fetch all users in the database.
POST	/api/add	Add new user to the database.
GET	/api/:id	Fetch user’s information by id.
PUT	/api/:id	Update user’s information.
DELETE	/api/:id	Delete user by id.
DELETE	/api/	Delete all user.

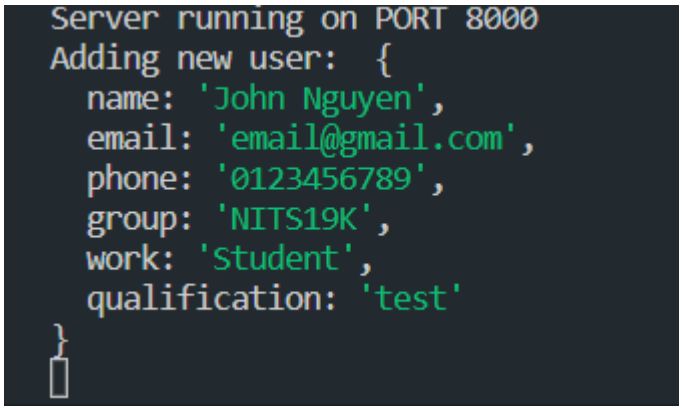
REST APIs were developed in the controller file in accordance with the TABLE 1 above to produce the desired outcome. The names of the functions as an explanation for themselves. By separating the routes and the controllers, the application code is more organized and easier to maintain. Each function

is responsible for a specific task, making it easier to debug and test the code. Additionally, this separation of concerns allows developers to make changes to one part of the application without affecting the other parts, resulting in a more modular and scalable codebase. PICTURE 22 presents some of the methods in the controller file.



PICTURE 22. A screenshot of controller file.

In the PICTURE 22, a function called “addUser” is created to add new user to the database, the new data comes to server from user which is request(req), will be used to create new instance of the User model and save the created user to the database. If the action is successful, the response will return an object which contains new user’s information and a successful message. The PICTURE 23 shows the new document added to the database from user’s input will be logged to the terminal.



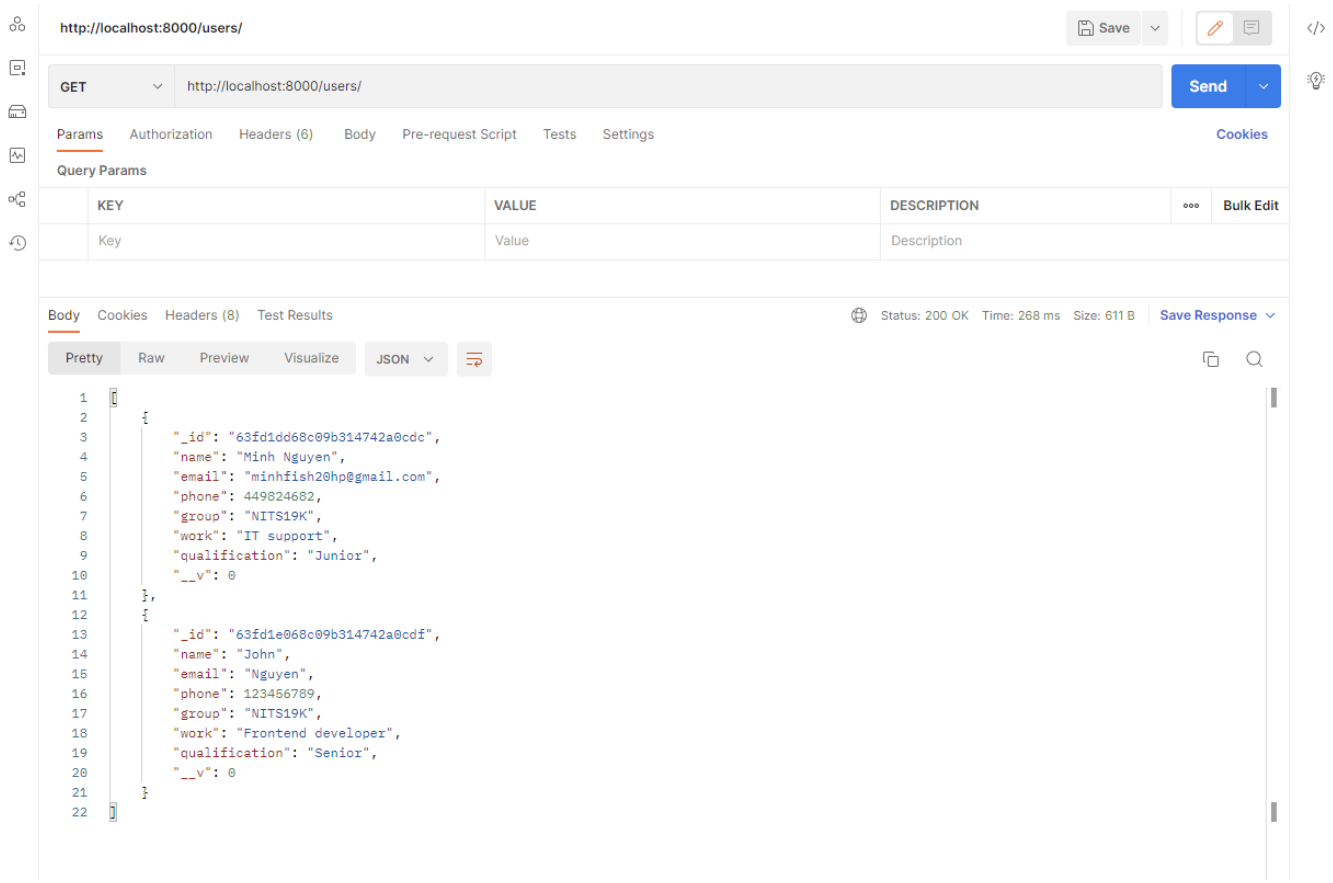
PICTURE 23. Adding new user to the database.

3.2.4 APIs testing with Postman

A client utility for integrating API testing is called Postman. API testing entails gathering APIs and determining whether they deliver the expected usefulness, dependability, speed, and security. API testing makes it possible to ascertain whether the product is well-organized and beneficial to other applications. The time it takes the API to obtain and approve the data is also calculated, along with a verification of the answer based on the request parameter. Additionally, it has a feature that allows testing of the same request across various settings and with various particular factors. By making a request to the web server and then receiving an answer, all of the application's APIs were tested in Postman. It operates on the server side and confirms that each API route is functioning properly. In postman, the user must make a request before postman checks the part the user is searching for in the response. The JS code that executes after a request has been sent and an answer has been received from the server is the target of Postman's test (Ojo, 2022).

In order to test an application's APIs, one must use the collection of API calls that Postman offers. The body content, headers, and status code make up an API answer. Requests like Add, Delete, and Update can transmit any necessary data, including criteria and permission information. Postman shows the answer from the API server after the call has been made so that the response can be examined, visualized, and if necessary, troubleshooted. The user has the option to save each answer to a request and refresh them as needed. The Postman body tabs have a number of instruments to aid in rapidly comprehending the answer. The raw view of the pretty model is a big text area with a response body that can be minified, and it organizes JSON and XML answers to make them simpler to read. The answers are also displayed in a sandboxed iframe on the preview page. Under the headers pane, headers are

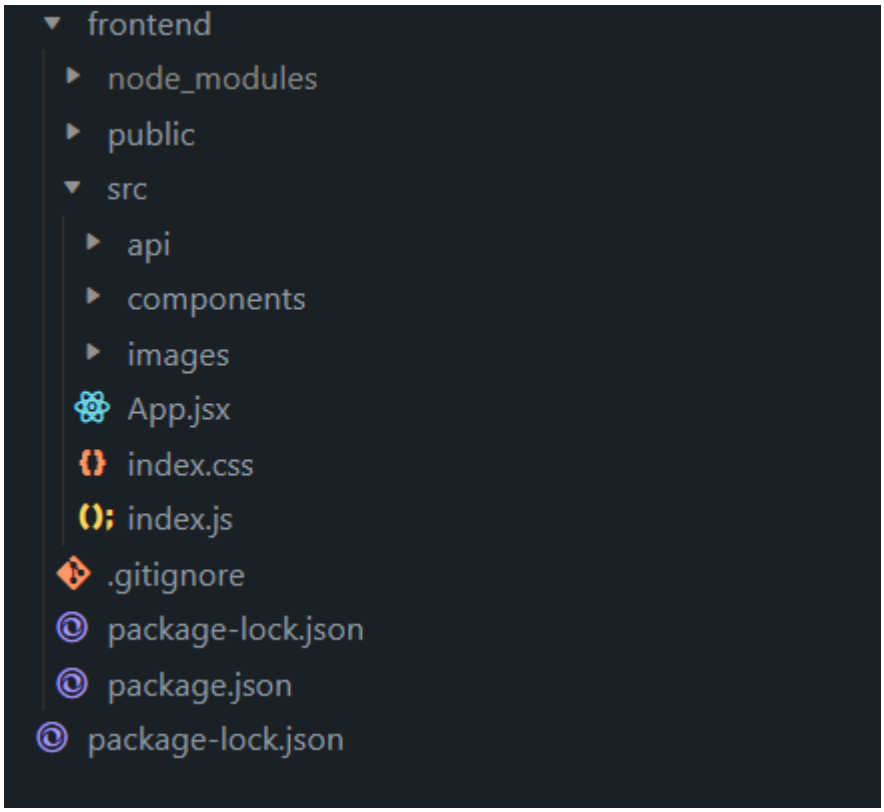
shown as key-value pairs with their descriptions as per the HTTP standard. Additionally, Postman divides the size of the replies into the approximative body and heading size. The PICTURE 24 is an example of using Postman to test API (Postman Learning Center, 2020).



PICTURE 24. Example of using Postman to test API.

3.3 Front-end Implementation

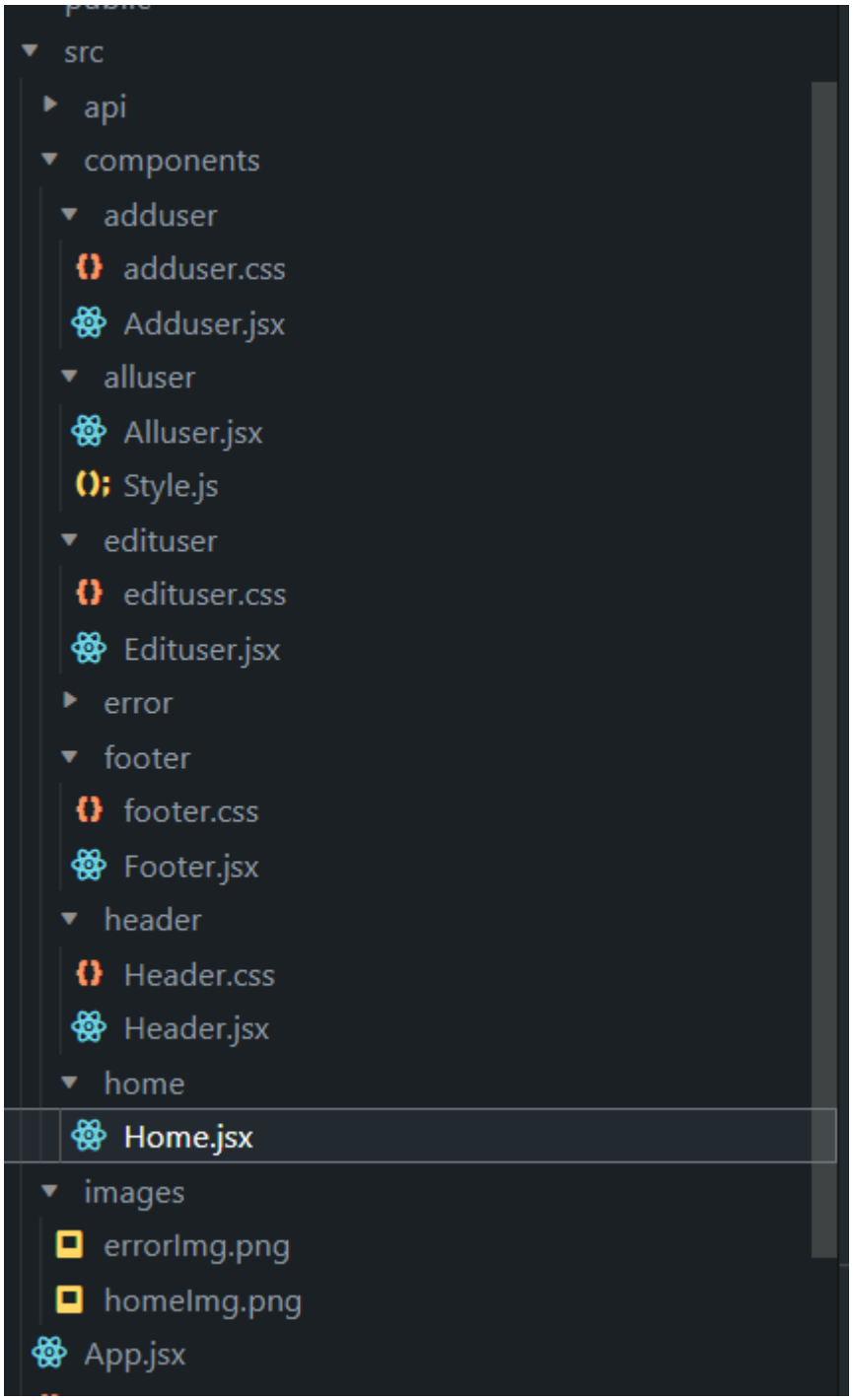
Starting a new React project used to be a complicated multi-step process, setup and environment settings were complex. Setting up Babel to transform JSX into browser-ready code and configuring Webpack to combine all the application files were both part of the procedure. Recognizing that this is a time-consuming and laborious process, Facebook created the "create-react-app" app Node.js package to rapidly build a React.js template. It includes code trans-piling, basic linting, testing, and build systems. It also includes a server with hot reloading that will refresh page as the code changes. Finally, it will create a structure for code directories and components as PICTURE 25 shows below (Joe Morgan, 2020).



PICTURE 25. Code structure that generated automatically by “create-react-app”.

3.3.1 React.js application setup

The "components" folder houses the vast bulk of UI and logic-related components in charge of different tasks. Everything in ReactJS is componentized because it strongly relies on component-driven programming, including the table, landing page, navbar, and form. The component-driven methodology speeds up development and makes it simple to move components around. The PICTURE 26 shows the components folder.



PICTURE 26. A screenshot of components folder.

In the src folder, “App.js” is the entry point and the main component that is responsible for rendering the entire application. It defines the structure and behaviour of the application’s main component, and it is usually the first component that gets rendered in the “index.js” file. This file typically contains the main layout of the application, including any navigation components as well as the routes of the application. PICTURE 27 presents the “App.js” file in this project.

App.jsx

frontend > src > App.jsx > App

```jsx
import React from "react";
import { Routes, Route } from "react-router-dom";
import Home from "./components/home/Home";
import Adduser from "./components/adduser/Adduser";
import Alluser from "./components/alluser/Alluser";
import Error from "./components/error/Error";
import Edituser from "./components/edituser/Edituser";
const App = () => {
  return (
    <>
      <Routes>
        <Route exact path="/" element={<Home />} />
        <Route exact path="/all" element={<Alluser />} />
        <Route exact path="/add" element={<Adduser />} />
        <Route exact path="/edit/:id" element={<Edituser />} />
        <Route exact path="*" element={<Error />} />
      </Routes>
    </>
  );
};

export default App;

```

PICTURE 27. A screenshot of the “App.js” file.

“react-router-dom” is installed in in project in order to define the navigation and routing logic of a single-page application. The “Routes” component serves as the container for all routes, while the “Route” component represents a single route. Each “Route” component takes two main props, “path” and “element”. The “path” prop specifies the URL path of the route, while the “element” prop specifies the component to be rendered when the route is visited. In this PICTURE 27, there are five routes specified. Each route corresponds to each component exported from the components folder.

3.3.2 API handling and data fetching

API handling and data fetching are important concepts in React applications that involve fetching data from a server-side API and using that data to render the user interface. API handling refers to the pro-

cess of communicating with a server-side API and exchanging data with it. This involves making HTTP requests to the API, processing the response data, and handling any errors that may occur during the process.

In React, API handling is typically done using the “fetch()” function or a third-party library like Axios or jQuery. The “fetch()” function is a built-in JavaScript function that makes HTTP requests and returns a promise that resolves to the response object. The response object can be converted to JSON format and used to update the application state. Data fetching refers to the process of retrieving data from an API and using that data to update the application state. This involves defining the API endpoint and making an HTTP request to fetch the data. Once the data is retrieved, it can be used to update the application state, which in turn triggers a re-rendering of the user interface.

In this project, Axios library is installed to handle this. “api” folder is created to handle API calling as the PICTURE 28 shows below. In this file, five functions are implemented, which function has their own parameters to send to URL endpoints to get data from back-end server. After that, all functions are exported so they can be used in other components. For example, in the PICTURE 29, the “getUsers” function will get all the users that exist in the database and display in the user interface. The function is call in the “useEffect()” hook in the “Alluser.jsx” file and the state will be updated as the data received from the function.

index.js

frontend > src > api > index.js > ...

```
1   import axios from "axios";
2
3   const usersUrl = "http://localhost:8000/users";
4
5   export const getUsers = async (id) => {
6     id = id || "";
7     return await axios.get(`${usersUrl}/${id}`);
8   };
9
10  export const addUser = async (user) => {
11    return await axios.post(`${usersUrl}/add`, user);
12  };
13
14  export const deleteUser = async (id) => {
15    return await axios.delete(`${usersUrl}/${id}`);
16  };
17
18  export const editUser = async (id, user) => {
19    return await axios.put(`${usersUrl}/${id}`, user);
20  };
21
22  export const deleteAllUser = async (user) => {
23    return await axios.delete(`${usersUrl}`, user);
24  };
25
```

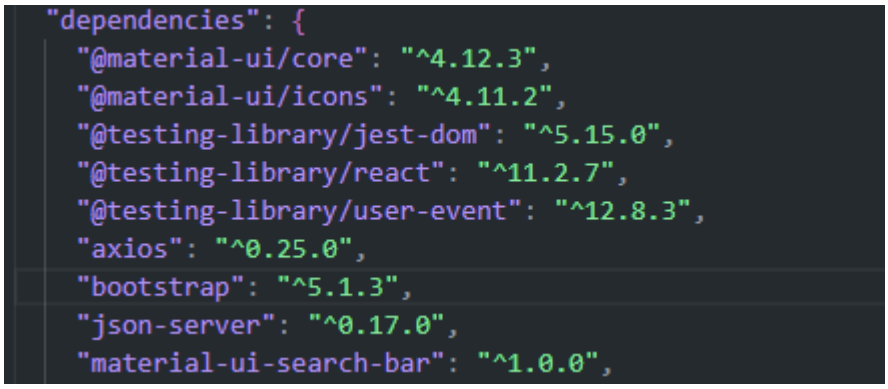
PICTURE 28. API handling file.

```
const Alluser = () => {
  const [users, setUsers] = useState([]);
  const classes = useStyles();
  const navigate = useNavigate();

  // Search Functionality Start //
  const [searchTerm, setsearchTerm] = useState("");
  // Search Functionality End//

  useEffect(() => {
    getAllUsers();
  }, []);

  const getAllUsers = async () => {
    let res = await getUsers();
    setUsers(res.data);
  };
```

PICTURE 29. An example of API handling and data fetching in the project.

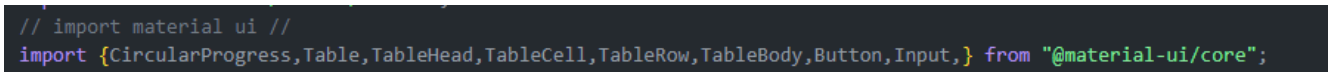
3.3.3 User Interface

In this project, Material UI and Boostrap are installed to build user interface of the application. Material UI and Bootstrap are both popular front-end development frameworks used to build modern, responsive and user-friendly web applications. Material UI is a React-based framework that provides a set of customizable components, icons, and themes, inspired by Google's Material Design language. It offers a clean and intuitive user interface and is widely used in building complex and scalable web applications. Bootstrap is a popular open-source front-end framework developed by Twitter. It provides a collection of CSS and JavaScript components, including grids, forms, buttons, and typography. Bootstrap is designed to simplify the process of building responsive and mobile-first web applications, and it has a large community of developers that contribute to its development and maintenance. PICTURE 30 shows the vesion of material-ui and bootstrap are being used in the project.



PICTURE 30. Material-UI and Bootstrap vesions.

To use Material UI components in React after the library is installed, components can be imported in the React application and add it to React component's render method. The PICTURE 31 and PICTURE 32 show an example of using Material UI components in React file. Components can be customized the appearance and behavior by passing props to them, as shown in PICTURE 32. The "variant" prop changes the button style, and the "color" prop changes the button color.



PICTURE 31. Import Material UI components into React file.

```
})
.map((user) => (
  <TableRow className={classes.row} key={user._id}>
    <TableCell>{user._id}</TableCell>
    {/* change it to user.id to use JSON Server */}
    <TableCell>{user.name}</TableCell>
    <TableCell>{user.email}</TableCell>
    <TableCell>{user.phone}</TableCell>
    <TableCell>{user.group}</TableCell>
    <TableCell>{user.work}</TableCell>
    <TableCell>{user.qualification}</TableCell>

    <TableCell>
      <Button
        title="Edit User"
        color="primary"
        variant="contained"
        style={{ marginRight: 10 }}
        component={Link}
        to={`/edit/${user._id}`}
      >
        <i className="fas fa-edit"></i>
      </Button>
      {/* change it to user.id to use JSON Server */}
      <Button
        title="Delete User"
        color="secondary"
        variant="contained"
        onClick={() => deleteUserData(user._id)}
      >
        <i className="fas fa-trash-alt"></i>
      </Button>
      {/* change it to user.id to use JSON Server */}
    </TableCell>
  </TableRow>
```

PICTURE 32. Components is used in React file.

4 CONCLUSION

In conclusion, the development of a simple CRUD application for user management is an important task for any modern organization. This thesis has presented the design, development, and evaluation of such an application, highlighting the benefits of using modern technologies such as React, Node.js, and MongoDB. The application was developed using a user-centered design approach, which involved understanding the needs of the users and designing the application accordingly. The application allows users to perform basic CRUD operations on user data, including creating, reading, updating, and deleting user records. Through the development and evaluation of the application, it was found that modern technologies such as React and Material UI can greatly enhance the user experience of the application. The use of Node.js and MongoDB also provided a scalable and efficient platform for data storage and retrieval.

Overall, the development of this simple CRUD application for user management demonstrates the importance of user-centered design and the benefits of using modern technologies in application development. The application provides a solid foundation for further development and can be used as a starting point for similar applications in the future.

REFERENCES

Altamira team, 2020. Best Web Application Technology Stack in 2022. Available at: https://www.altamira.ai/blog/how-to-choose-your-technology-stack/. Accessed 12 February 2023.

Bennison J, 2022. What is module in node js. Available at: https://medium.com/yavar/what-is-the-node-module-in-node-js-19ef41820af8. Accessed 13 February 2023.

Blischak J, Davenport E, Wilson G, 2016. A quick introduction to Version Control with Git and Github. Available at: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004668. Accessed 14 February 2023.

Bormon M, 2022. What is Mongoose? Available at: https://medium.com/@monibbormon14/what-is-mongoose-c1bc3031cc08. Accessed at 13 February 2023.

Code, 2020. Documentation for Visual Studio Code. Available at: : https://code.visualstudio.com/docs. Accessed 20 February 2023.

Chaddha M, 2022. Routing in Node.js. Available at: https://www.codingninjas.com/codestudio/library/routing-in-node-js. Accessed 13 February 2023.

Educative, Inc, 2022. What is mern stack. Available at: https://www.educative.io/answers/what-is-mern-stack. Accessed 12 February 2023.

Flexible, 2023. An Intro to ExpressJS. Available at: https://flexiple.com/express-js/deep-dive/. Accessed 12 February 2023.

Javapoint, 2022. React Component. Available at: https://www.javatpoint.com/react-components. Accessed 12 February 2023.

Hamedani M, 2018. React functional components. Available at: https://programmingwithmosh.com/react/react-functional-components/. Accessed 12 February 2023.

Heller M, 2022. What is Node.js? The Javascript runtime explained. Available at: https://www.infoworld.com/article/3210589/what-is-nodejs-javascript-runtime-explained.html. Accessed 13 February 2023.

Kolesnikova T, 2022. Event-driven development features. Available at: https://studybay.com/blog/event-driven-development-features/. Accessed 13 February 2023.

Morgan J, 2020. How to set up a React project with Create React App. Available at: https://www.digitalocean.com/community/tutorials/how-to-set-up-a-react-project-with-create-react-app. Accessed 14 February 2023.

Ojo M, 2022. How to automate API tests with Postman. Available at: https://blog.logrocket.com/how-automate-api-tests-postman/. Accessed 14 February 2023.

Mojeed I, 2022. What is the Virtual DOM in React. Available at: https://blog.logrocket.com/virtual-dom-react/ . Accessed 12 February 2023.

Npm trends, 2022. Angular vs React vs Vue. Available at: https://npmtrends.com/angular-vs-react-vs-vue. Accessed 12 February 2023.

Pandit N, 2021. What and why React.js. Available at: https://www.c-sharpcorner.com/article/what-and-why-reactjs/. Accessed 12 February 2023.

Postman Learning Center, 2020. Available at: https://learning.postman.com/docs/sending-requests/requests/. Accessed 20 February 2023.

Selvaganesh, 2018. How Nodejs middleware works? Available at: https://selvaganesh93.medium.com/how-node-js-middleware-works-d8e02a936113. Accessed 12 February 2023.

Sonia M, 2021. What is NPM? Available at: https://www.codementor.io/@soniaiom1705/what-is-npm-node-package-manager-1h9jsc2515. Accessed 20 February 2023.

Sufiyan T, 2022. Understanding Node.js Architecture. Available at: https://www.simplilearn.com/understanding-node-js-architecture-article. Accessed 12 February 2023.

Tutorialpoints, 2023. Component-Based Architecture. Available at: https://www.tutorialspoint.com/software_architecture_design/component_based_architecture.htm . Accessed 12 February 2023.

Vaughan J, 2022. MongoDb. Available at: https://www.techtarget.com/searchdatamanagement/definition/MongoDB. Accessed 12 February 2023.

Vishnupriya, 2022. What are React JS Class Component? Available at: https://codedamn.com/news/reactjs/what-are-react-js-class-components. Accessed 12 February 2023.

Zaig R, 2019, The NodeJs Event Loop. Available at: https://medium.com/payu-engineering/the-node-js-event-loop-ecde345dbc57. Accessed 13 February 2023.