Social Media Platform

A

Project Report

submitted

in partial fulfilment

for the award of the Degree of

Bachelor of Technology

in

Department of Computer Science & Engineering

Supervisor:

Ila Sharma

Assistant Professor

(Computer Science & Engineering Department)

Submitted By:

Noor Fatema Neelgar

21EMICS042

Tanu Shree Kachhawa

21EMICS060

Jitendra Verma

21EMICS027

Department of Computer Science & Engineering

Modi Institute of Technology

Rajasthan Technical University

April, 2025

(Session: 2024-25)

Candidate's Declaration

I hereby declare that the work, which is being presented in the Major Project, entitled "Social Media Platform" in partial fulfilment for the award of Degree of "Bachelor of Technology" in Deptt. of Computer Engineering with Specialization in Computer Science and Engineering and submitted to the Department of Computer Science and Engineering**,** Modi Institute of Technology, Rajasthan Technical University is a record of my own investigations carried under the Guidance of Ila Sharma, Assistant Professor Department of Computer Science and Engineering, Modi Institute of Technology**.**

I have not submitted the matter presented in this report anywhere for the award of any other Degree.

Noor Fatema Neelgar

Computer Science and Engineering

Roll No.: 21EMICS042

Tanu Shree Kachhawa

Computer Science and Engineering

Roll No.: 21EMICS060

Jitendra Verma

Computer Science and Engineering

Roll No.: 21EMICS027

Modi Institute of Technology, Kota

Counter Signed By:

Ila Sharma

Assistant Professor, CSE Dept.

CERTIFICATE

This is to certify that Noor Fatema Neelgar, Tanu Shree Kachhawa, Jitendra Verma of VIII Semester, B.Tech (Computer Science & Engineering) 2024-25, has presented a major project seminar titled "Social Media Platform" in partial fulfilment for the award of the degree of Bachelor of Technology under Rajasthan Technical University, Kota.

Date:

Ila Sharma	Mrs. Seema Arya
Assistant Professor, CSE Dept	H.O.D.

Place: Kota

ACKNOWLEDGMENET

I take this opportunity to express my gratitude to all those people who have been directly and indirectly with me during the competition of this project.
I pay thank to Ila Sharma who has given guidance and a light to me during this major project. Her versatile knowledge about “Social Media Platform” has eased me in the critical times during the span of this major project.
I acknowledge here out debt to those who contributed significantly to one or more steps. I take full responsibility for any remaining sins of omission and commission.

Noor Fatema Neelgar
Tanu Shree Kachhawa
Jitendra Verma
B.Tech IV Year
(Computer Science & Engineering)

CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Introduction

It is a full stack responsive social media application using MERN stack that implements CRUD operations based on the three layered architecture. Programs at each layer have their own unit test. The social media application consists of a register page with complete validation along with functionality to upload a user image for their profile. Any user can register and can then use the registered email address and password to sign in. The home page has a very clean UI with a number of different widgets. There is also a box that shows the complete user profile details. The current user that's signed in will be able to make posts as well as add an image for the post. They can edit, delete and actually make the post. They can also see the users news feed of all the posts that have been created. The users can also actually like and dislike any posts that they want. They can view the comments as well. They can add a friend if they would like and the friend list will be updated over. The users can add any amount of friends and can remove them if we want to. They can also view the profiles of other users and can see the other person’s friends. The person also has the ability to write a post for another user and then can see their user post as well. We can change from light mode to dark mode and vice versa and more importantly everything is going to be completely fully responsive so the user can see the exact same website on smaller screens with modified adjustments for everything most importantly. Everything on the frontend, all the information comes from backend APIs which are retrieving information from the databases(MongoDB). The project is done on the MERN stack which includes Mongo database, Express.js, React and Node. Specifically for the front end, React is used as a framework. React router for navigation. Formic and YUP for form and form validation. Redux toolkit for state management with Redux Persist to store in local storage and React Dropzone for image uploads.

For the back end, Node.js is used as the runtime environment. Express.js as the backend framework. Mongoose for managing the database. JSON Web Token (JWT) for authentication and Multer for file uploading.

There is also an implementation of middleware that authenticates the http request before sending it to the server. Front-end of the website is made using html, css, javascript.

Bootstrap templates are used extensively. For the back-end we are using nodejs and its packages such as Express js, JWT(for authentication and security). Its library Mongoose is used to write more readable code.

1.2 Problem Statement

To apply industrial best practices and create a fast, scalable and secure web application. To learn and apply the knowledge of front-end development in real life projects and to understand the in-depth working of MERN Stack applications.

1.3 Objectives

To create testable, structured, clean and maintainable web applications by using industrial best practices. To apply the knowledge about the technologies thought to us thus far and gain practical experience.

1.4 Methodology

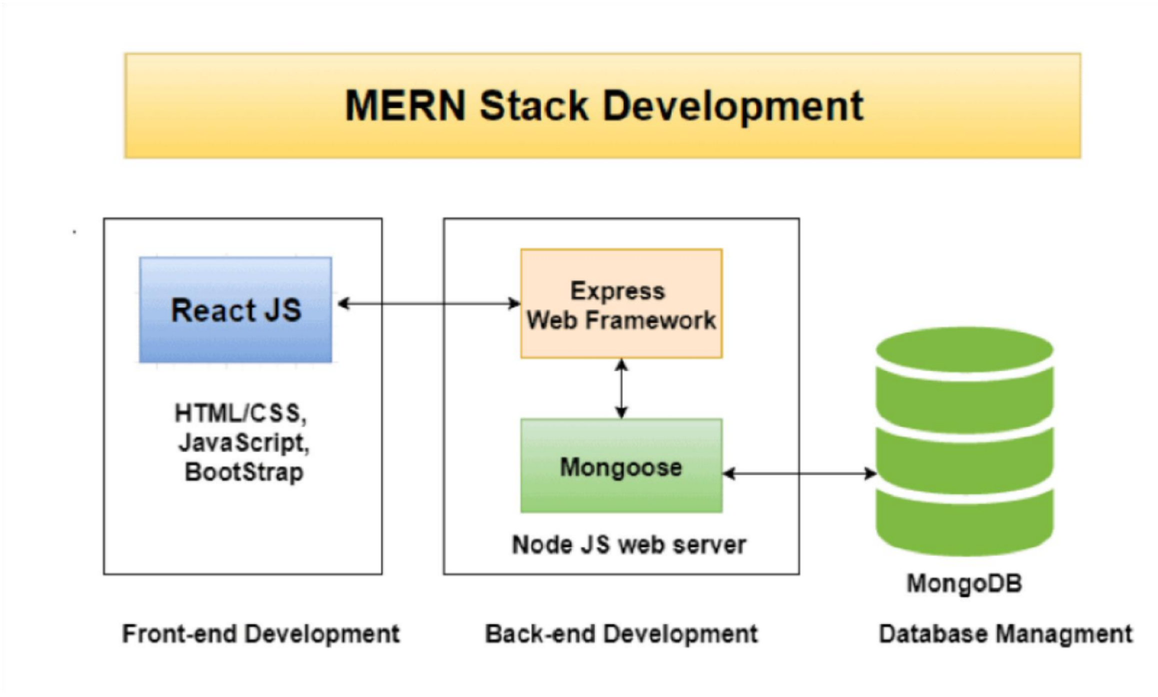


Fig 1.1 MERN Stack Architecture

Chapter 2

Literature Survey

Full-stack developers are in greater demand than ever before in the modern world. The biggest demand is accompanied by an amazing average income of $110,770 in the US, according to a poll conducted by Indeed. [1] A person who is technically capable of working on the front-end and back-end development of a dynamic website or web-based apps is referred to as a "full-stack developer."

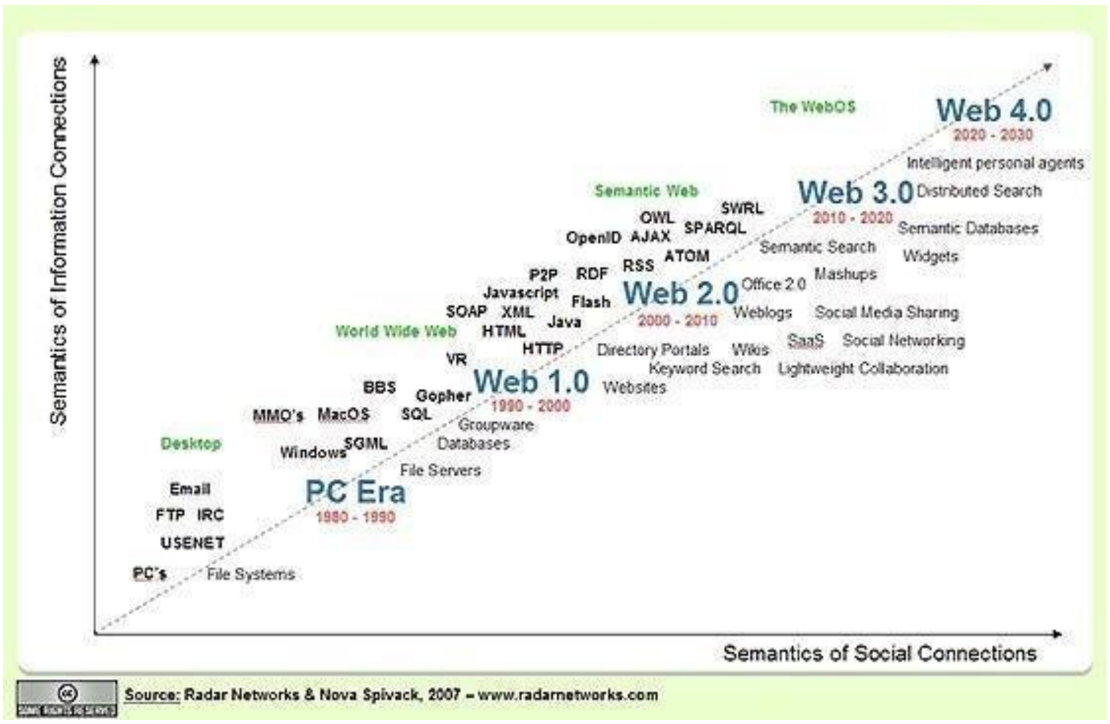


Fig 2.1 The Evolution of Web

The foundations of the modern commercial internet were laid in 1990. Tim Berners-Lee developed the fundamental ideas of the World Wide Web as well as other tools for effective web usage at the end of 1990.

These include the HyperText Transfer Protocol (HTTP), the HyperText Markup Language (HTML), the first web browser and code editor, the first web server, and the first web page that introduced the concept of the world wide web as well as a technique for creating one's own web page [8]. Since 1990, the internet has rapidly developed, and four generations of development may be identified (Fig. 2.1)[11]. Users could only browse web material on the first generation's static, infrequently updated web sites.

New technologies that enable the presentation and delivery of web services without issues with web distribution also emerged at that time, including JavaScript, Document Object Model (DOM), Ajax, Cascading Style Sheets (CSS), eXtensible HTML (XHTML), eXtensible Markup Language (XML), eXtensible Stylesheet Language (XSL), and Flash. 2010 marks the beginning of the third web generation, which is characterised by the semantic web (which adds semantics to the web), content personalisation, intelligent search, and computers' ability to create a variety of material. Ontologies are employed in the representation and justification of meaning. In addition to ontologies, technologies like Web Ontology Language (OWL), Resource Description Framework (RDF), and others are employed in the third generation of the web.

The LAMP stack, which consists of Linux, Apache, MySQL, PHP or Perl, and Java (Java EE, Spring), which includes a variety of programming languages, was the major foundation for web development in the past. With the advent of the MERN stack, JavaScript facilitates web development by having the ability to operate on both the clientand server-side. There are four main technologies in the stack: MongoDB, Express.js, React.js, and Node.js. Studying the nature of each component in the stack and developing a social platform that can link individuals were the main objectives of this project.The outcome is a platform that has sufficient features to demonstrate the connections of each of the components in the MERN stack.

The goal of the project is to implement the fundamental elements of the MERN Stack[4,6] technology, including MongoDB, ExpressJS, ReactJS, and NodeJS platform. Using the fundamental features of an e-commerce web application, such as sign-up and sign-in, dashboard display, and product and shop category display, building a web application with a payment gateway and product stores using MERN Stack technology. Implement website administration tools including user management, store management, analytics, and reporting.

Node.js is a system application, a server environment, and it is open source. Using the NodeJS platform, which was independently developed using JavaScript from Chrome, we can create network apps rapidly and easily. To run the code, use the JavaScript engine on Google. Additionally, a significant portion of necessary modules are written in JavaScript 6. Node.js includes a built-in framework that enables programmes to act as Web Servers similar to Apache HTTP Server.

Express.js A framework developed over NodeJS is called Express.js. It offers a wide range of cutting-edge features for web and mobile development. Because HTTP is supported by Express.js, the API is incredibly robust, dependable, and simple to use. Without slowing down NodeJS, Express adds more tools for developers that aid in creating a better programming environment[6]. The most popular NoSQL[6] database today, MongoDB, is free source and used by thousands of users. It was created using one of today's most widely used programming languages. Additionally, MongoDB is a cross-platform data store that utilises the notions of Documents and Collections, offering great performance with high availability and flexibility in terms of extension. Since this database was created using the JavaScript Framework and the JSON data type, it is a source database format that does not utilise Transact-SQL to access data[7]. With its introduction, it has been able to improve operating speed and functionality while overcoming the drawbacks of the RDBMS relational database management system concept.

Additionally, MongoDB is a cross-platform database that uses a collection- and document-based strategy to create sharp output, enormous availability, and simple scalability[11]. A scripting, object-oriented, and cross-platform programming language is JavaScript. Host environment objects can be linked to JavaScript and set up in a way that makes it operable. JavaScript includes common libraries like Array, Date, Math, and the core elements of programming languages including managers, control framework, and statements objects.

React is built around components. A component can be created by creating a Class function of the React object, the starting point of accessing this library. ReactJS creates HTML tags unlike we normally write but uses Component to wrap HTML tags into objects to render. Among React Components, render function is the most important.[9] It is a function that 104

Chapter 3
System Development

Architecture Design

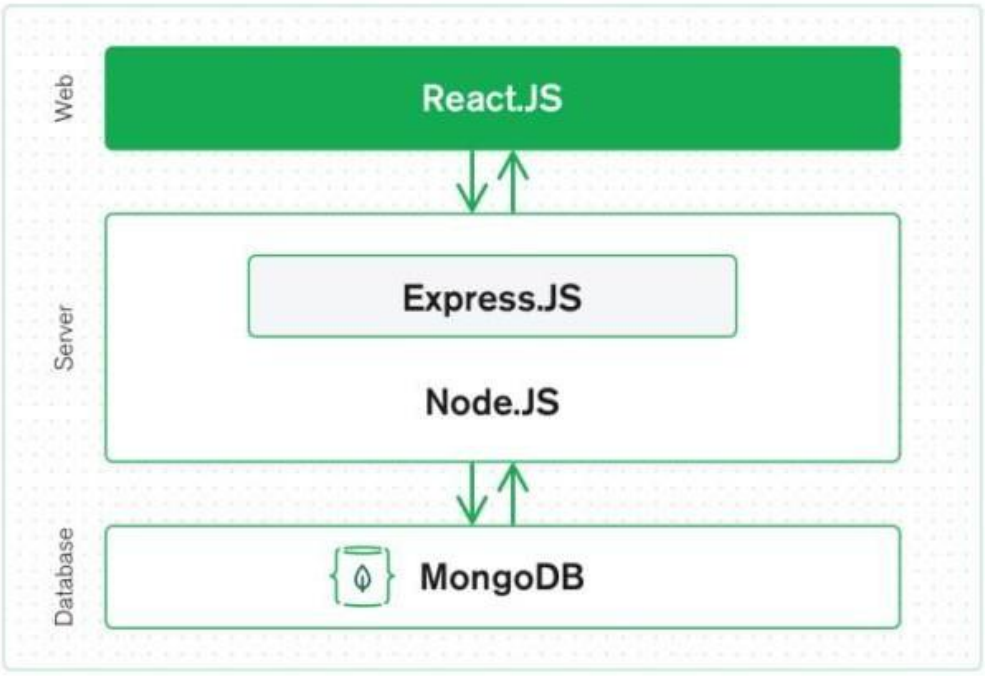


Fig 3.1 : A 3-tier MERN architecture

MERN, or MongoDB, Express, React, and Node.js, are acronyms. With MongoDB as the database, React.js is a web client library, Express.js is a web server framework, and Node.js is a server-side platform. It enables programmers to create Web applications that only use full-stack JavaScript.

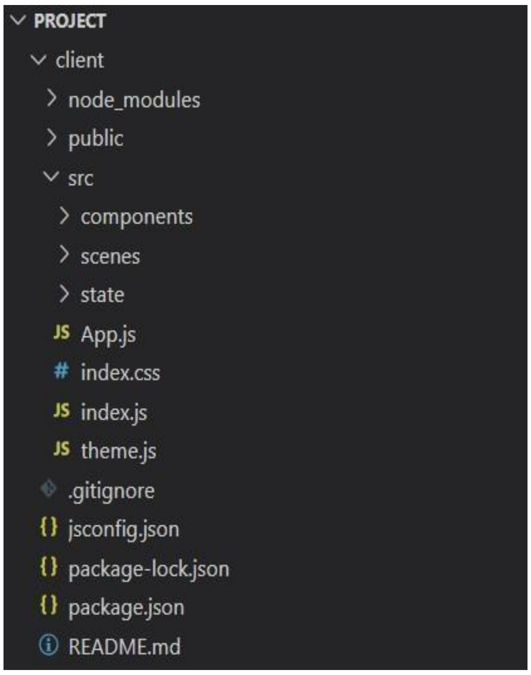
Since MERN combines four cutting-edge technologies, including Facebook's strong support, it eliminates the need for developers to learn other platforms like.NET, PHP, or Java. Learning new technologies for application development saves developers time and effort. The stack is supported by a large number of open-source packages and a committed community of programmers to boost scalability and maintain software due to the same JavaScript platform.

The foundation of the MERN stack is Node.js, which is a server-side technology with extremely high performance and quick response to all tasks, including massive and complex data, as shown in Fig 3.2. TypeScript is not required for MERN; all that is required is the adaptable React framework, which is now the most well-liked and influential front-end technology.

3.1 System design implementation

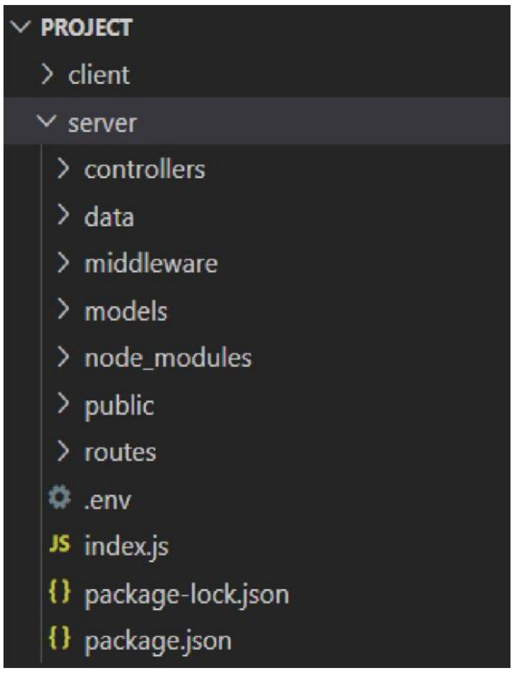
Separation of concerns : A react web application usually has two sub-folders for client-side and server-side applications. Each sub-folder is then divided into separate folders and files based on hierarchy.

The client side is mainly responsible for the user interface and experience while the creation, deletion, updation and retrieval of the data is managed in the server side folder.



```
PROJECT
  client
    node_modules
    public
    src
      components
      scenes
      state
      App.js
      index.css
      index.js
      theme.js
    .gitignore
    jsconfig.json
    package-lock.json
    package.json
    README.md
```

Fig 3.2 Client side work environment



```
PROJECT
  client
  server
    controllers
    data
    middleware
    models
    node_modules
    public
    routes
    .env
    index.js
    package-lock.json
    package.json
```

Fig 3.3 Server side work environment

3.1.1 Identification of features

The features of web application includes :

- Dark mode and Light Mode in UI - Creation of a post with images.
- Deletion of post

- Add Friends and view their profile
- Like and dislike a post
- Updation of a post only by a user who is logged-in.
- Deletion of an existing entry only by the user who created it.
- Upload Profile photo
- Creating a user (signup).
- Letting the user Login and Logout

3.1.2 Libraries and frameworks used Javascript :

An easy-to-use, interpreted, cross-platform scripting language is called JavaScript. In the past, JavaScript was solely used on websites and was executed by browsers to enhance the user experience. However, JavaScript can also be utilised to execute on the server, just as Node.js has been around since 2009 and uses the V8 engine.

• Client-side JavaScript: An addition to the fundamental JavaScript language that allows for browser and DOM management. Without regularly refreshing the page, it can also conduct some calculations, alter UI components, and validate input.

• Server-side JavaScript: An extension of the basic JavaScript language that may be viewed as regular C#, C, Python, or any other server-side programming language code and is designed to execute on a server. The second significant update to JavaScript was ECMAScript 6, commonly referred to as ES6 or ECMAScript 2015, which was published in 2016. [5] When implementing the React.js framework in any of the MERN-based projects, comprehension of ES6 is crucial. Developers will write standard JavaScript in React.js combined with capabilities from ES6.

Reactjs :

React is an open-source, free front-end library based on javascript used for creating component-based user interfaces. It is kept up-to-date by Meta and a group of independent programmers and businesses. With frameworks like Next.js, React may be used to create single-page, mobile, or server-rendered applications. Routing and other client-side functionality are frequently provided by libraries in React applications because React is solely concerned with the user interface and rendering components to the DOM. React.JS is used to create single-page applications because it can render dynamically changing data quickly, and was used to design our web application's user interface. Developers can construct User Interface components using React and JS coding. We researched Reactjs virtual DOM objects before using them in our project. Any modifications we made to our web application for online shopping made the entire user interface render the virtual DOM again. The potential differences between the DOM Object and Virtual DOM can then be compared thanks to this. We used JSX, which made writing our code for the React

application simpler and easier.[4]

Components are used by React.JS. The foundation of User-Interface is made up of components, each of which contributes to the overall User-Interface of our web application and has logic connected to our social media application. Reusing components made our web application code simpler for other developers to understand and improved the efficiency of the web application as a whole.

Installing create react-app using npm or yarn was the first step in starting our React application. npm install yarn global adds OR create-react-app global The two commands for using npm or yarn are create-react-app.

Nodejs :

Node.js is a JS operating system that was created in the C++ programming language. A JS runtime environment is Node.js. For fast performance, Node.js makes advantage of the Google Chrome V8 engine. Node.js design uses the event-driven as the fundamental core concept for its environment, which gave us the various number of APIs that are event-based and asynchronous in nature which has helped us in building the website using node.js for our back-end development.[2] As we used Node.js, it used the corresponding callback function according to our web application’s business logic. These callback functions are executed asynchronously, which means that although these functions appear to be registered sequentially in the logic, they do not depend on the code written in which they appear, but rather wait for the execution of the corresponding event to fire. The main advantage of Event-driven and asynchronous programming is that it uses single-threaded architecture[1].

The restricted resources were used for other tasks that needed to be completed as part of the business logic for our web apps while the call back function code was still being executed. This layout suited our back-end development, which was another objective of our system. In server development, responding to synchronous requests was a challenging operation, and blocking played a big role in underutilizing or squandering resources. We enhanced the resource use and optimised the performance of our website using single thread architecture and asynchronous callback mechanisms, which also provided us with the necessary testing outcomes.

We can see that many of the functions, including file operations, are done asynchronously, which is different from other languages, in the supported module that Node.js provides.

Node.js uses particularly big network modules, such as HTTP, DNS, NET, UDP, HTTPS, TLS, and others, to make server development easier. Developers can set up a Web server using these network modules.

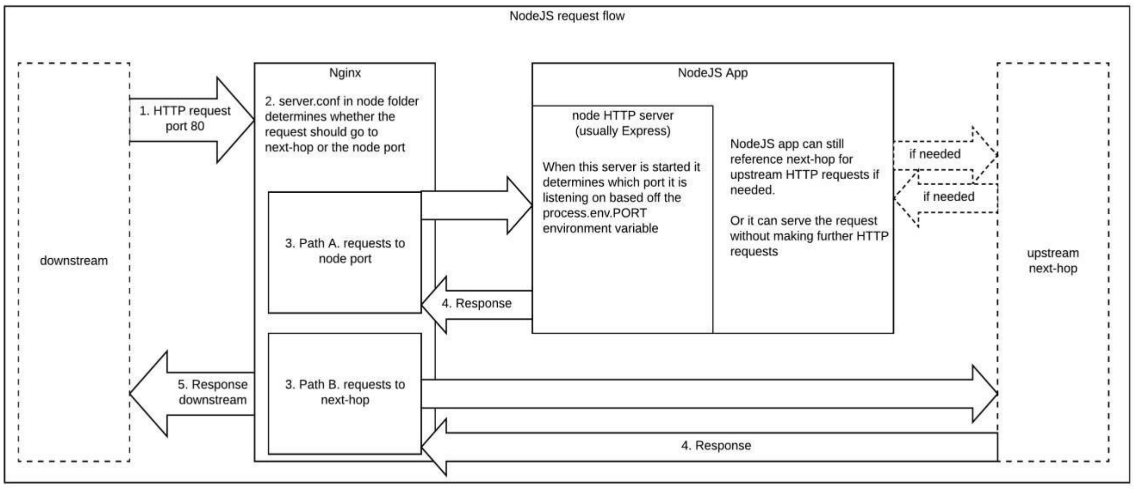

NodeJS request flow
downstream
1. HTTP request port 80
Nginx
2. server.conf in node folder determines whether the request should go to next-hop or the node port
3. Path A. requests to node port
4. Response
3. Path B. requests to next-hop
5. Response downstream
4. Response
NodeJS App
node HTTP server (usually Express)
When this server is started it determines which port it is listening on based off the process.env.PORT environment variable
NodeJS app can still reference next-hop for upstream HTTP requests if needed.
Or it can serve the request without making further HTTP requests
if needed
if needed
upstream next-hop

Fig 3.4 NodeJS Request Flow

ExpressJS:

Express is a Node.js framework, thus we used it. While developing the application, we discovered that Express made it simpler and easier to create the back-end code and implement it in an organised style rather than creating a tonne of node modules and writing the code with NodeJS.[3] We used Express to develop the web applications and APIs needed for our project because it supports a variety of middlewares that make coding quicker and simpler. The two main benefits of adopting Express in our application are asynchronous programming and single-threaded architecture. comprehensive API for our application We had to add a command to the command prompt to initialise the package after creating a new folder to begin our express project.json file. After that, we had to accept the default settings and continue. npm init is the command to start.

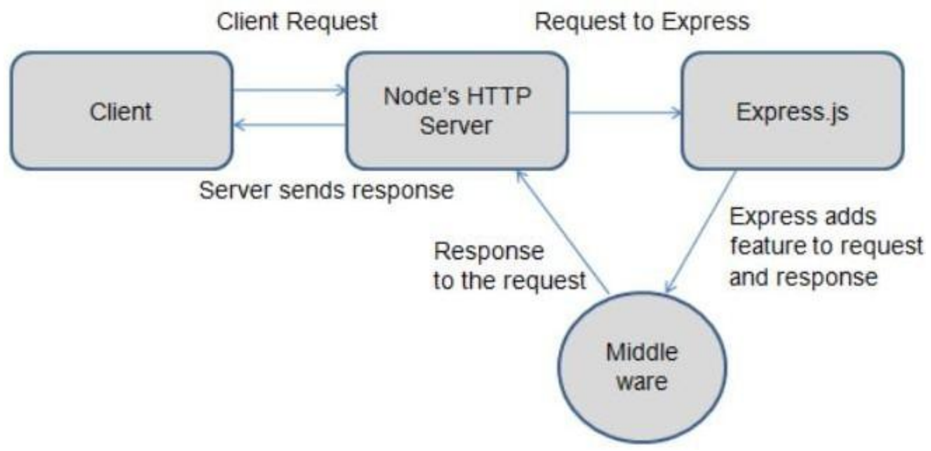


Fig 3.5 Express Js Request Flow

Mongoose :

Mongoose is a mongodb library for writing concise and readable code. It handles data associations, offers validation, and translates between objects created in code and how MongoDB represents those same items. This indicates that Mongoose enables the definition of objects with strongly-typed schemas that map to MongoDB documents. Mongoose offers a staggering amount of capabilities for developing and interacting with schemas. CRUD activities that are challenging to carry out with raw MongoDB can be carried out quickly and effectively using Mongoose.

MongoDB:

For our project, we chose the document-oriented database MongoDB. Every record in the MongoDB database is a document format. MongoDB transforms our JSON data into a binary version in the background on the server, which can then be stored and queried more quickly. MongoDB employs BSON for database queries. Although we can't think about

MongoDB as a JSON database because it saves BSON format both internally and across the network, it is a database nonetheless. Any data that can be natively stored in MongoDB and then simply accessed in JSON format can be represented in JSON.

We may state that MongoDB is flexible and enables users to design schema, databases, tables, etc. after studying and implementing it. After installing MongoDB, we had the choice to use Mongo shell because it provides a JavaScript interface via which users can communicate and perform any query-related tasks. Since MongoDB is a document-oriented database, indexing documents is simple. and for that reason it manages answers more quickly. Scalability of MongoDB We managed massive amounts of data in the MongoDB database by splitting it out into a nested described structure. A database server called MongoDB enables us to run several databases on it. The ER relationship of all the tables is shown in Fig 3.6.

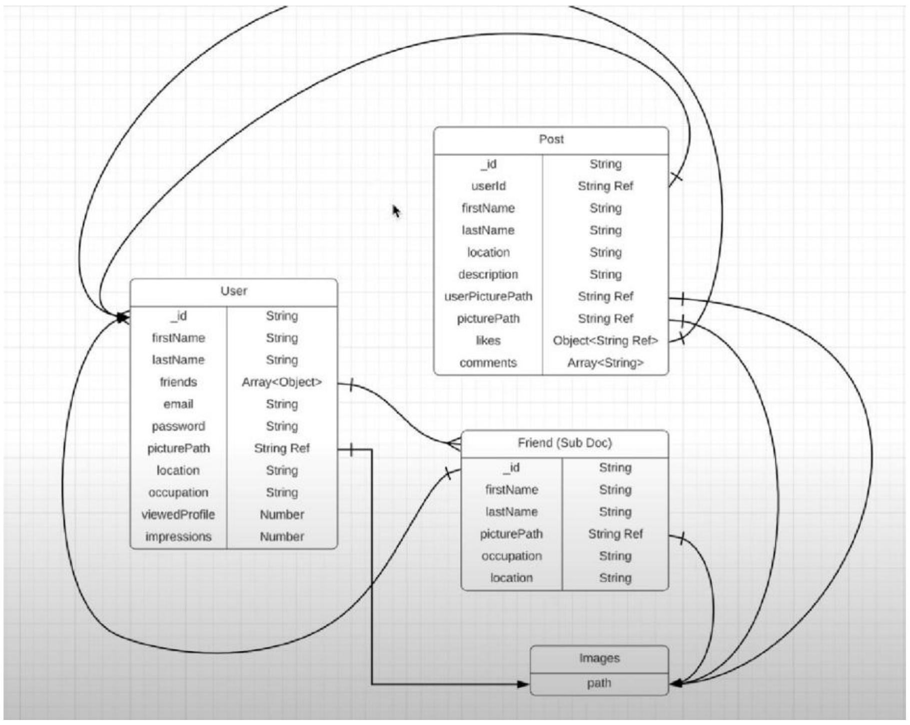


Fig 3.6 ER relationship of the MongoDB Database

Nodemon : If we make changes in the file and save it, nodemon starts the server automatically. Without nodemon one has to restart the server automatically after every change. It saves a lot of time and effort. While testing, the website can be run on localhost using the “**nodemon app**” command.

3.1.3 Technical Requirements

- **VS Code**(preferred IDE) / atom
- **Postman** api platform for building and testing APIs.
- **MongoDB** (Nosql) database.

3.2 Website Design

3.2.1 Registration Page

When the user first signs up, he will be directed to the registration page to create an account. They can enter their Firstname and Lastname. Their location and occupation. There is also a functionality to upload an image from the local file system. The user can then enter their email and password which will be stored in the backend to confirm their authentication and then login from the login page from then on. The registration page has complete validation. Fig 3.7 shows a screenshot of the registration page.

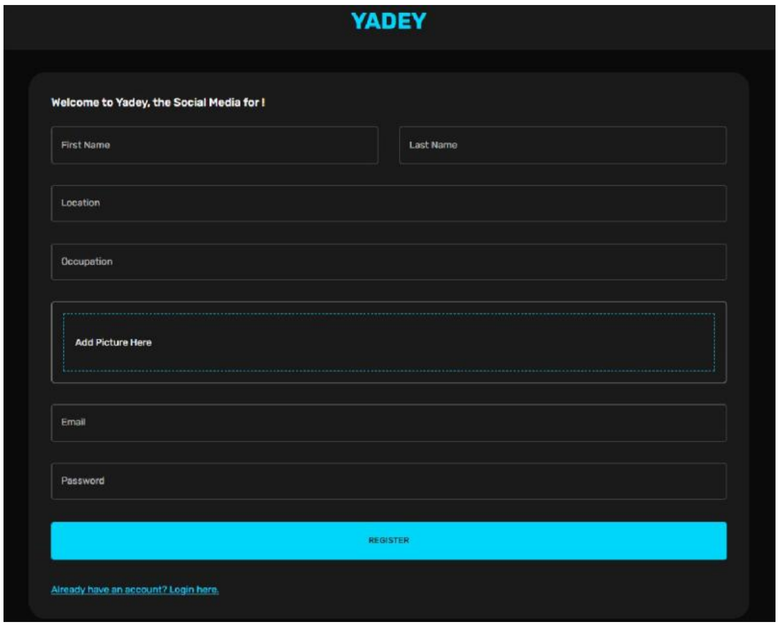


Fig 3.7 Registration page

3.3.1 Login Page

The user can login using the registered email address and password. The login page is shown in Fig 3.8

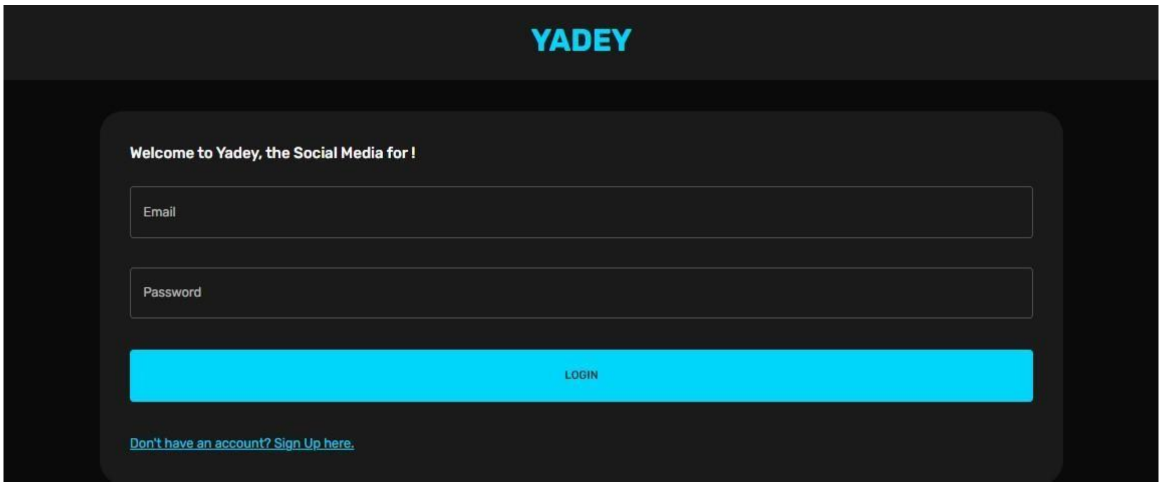
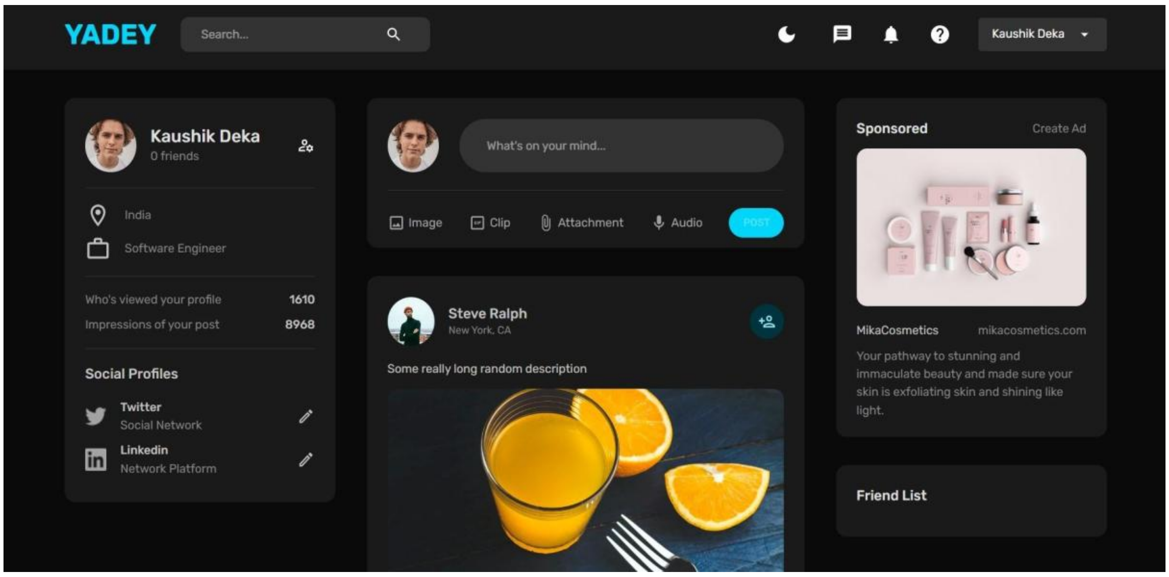

Welcome to Yadey, the Social Media for !

Fig 3.8 Login Page

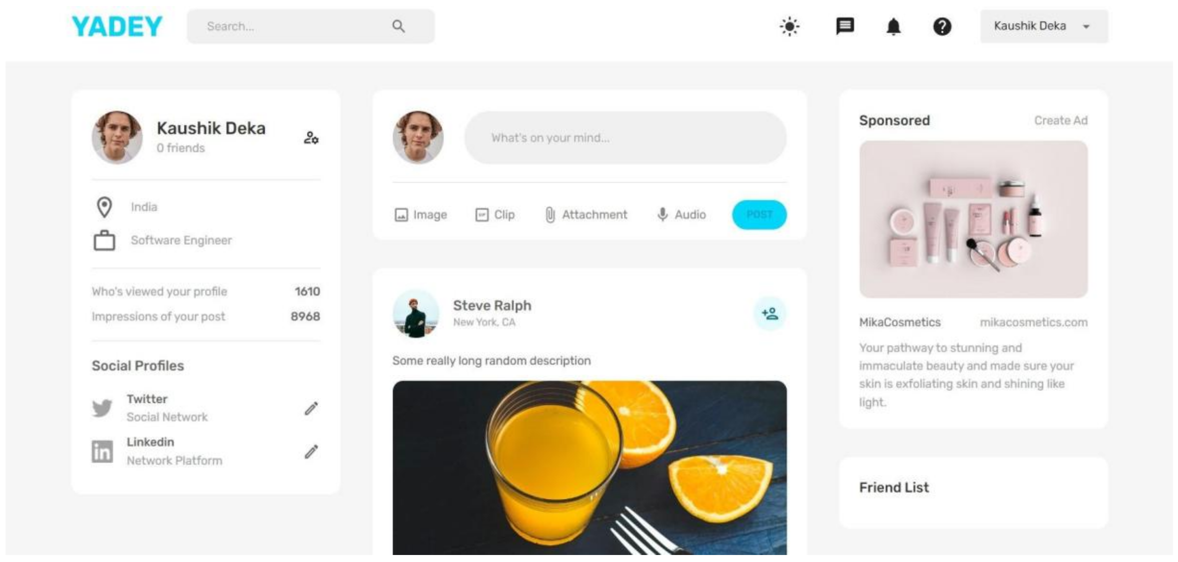
3.3.2 Home Page

The Homepage consists of the option to change between dark mode and light mode. It consists of all the posts by different users. The user can upload posts with pictures. Like and unlike posts. All the user details are shown in the homepage along with ads and friend list.


Some really long random description

Your pathway to stunning and immaculate beauty and made sure your skin is exfoliating skin and shining like light.

Fig 3.9 Home screen in Dark Mode



Some really long random description

Your pathway to stunning and immaculate beauty and made sure your skin is exfoliating skin and shining like light.

Fig 3.10 Home Screen in light mode

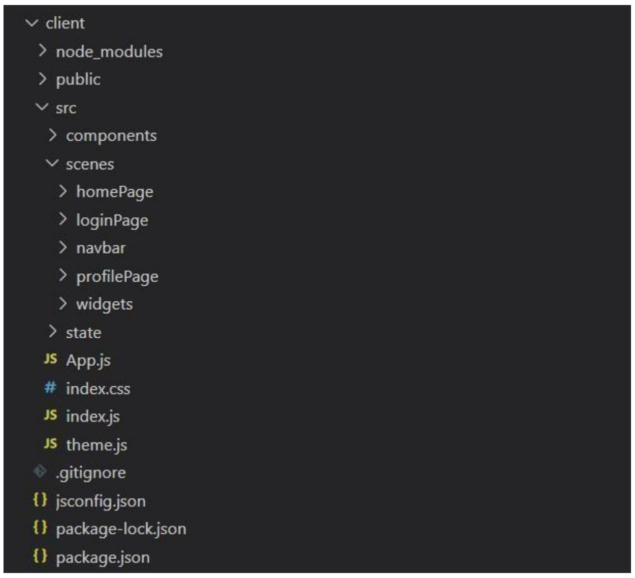
Fig 3.10 shows the UI in light mode and Fig 3.11 shows the UI in light mode.

3.3 Front-end implementation

3.3.1 Bootstrapping a basic react application

We need to run the command - npx create-react-app ***appname*** and it will automatically make a folder with all the requirements for creating a react application. Bootstrapped react app made using npm create-react-app command

3.3.2 React Workspace and folder hierarchy



```
client
  node_modules
  public
  src
    components
    scenes
      homePage
      loginPage
      navbar
      profilePage
      widgets
    state
    App.js
    index.css
    index.js
    theme.js
  .gitignore
  jsconfig.json
  package-lock.json
  package.json
```

Fig 3.11: React Workspace and folder hierarchy

Fig 3.11 shows how the folders and files are structured in the client directory. It is important to keep everything organised to avoid any mistakes.

3.3.3 Installing packages and dependencies

In react, various packages and dependencies can be installed using the following commands: npm install packagename - for installing dependencies normally. npm i - for installing all the dependencies in one go.

3.3.4 States in React

In react, any change made by the user is considered as a change in state. A state contains information about the component in which it is present. Whenever we change the state of a component, it renders again with a new state. The setState() constructor is used to change the state of a component. For example, if we type something in the search bar, with each letter the state is changing and the component has to re-render. Example :

```
Class MyClass extends React.Component
   { constructor(props) { super(props);
```

```
        this.state = { attribute : "value" };
    }
}
```

3.3.5 Props in react

Props is a shorthand notation for properties. It works similar to HTML attributes. A prop in react may seem similar to state but the major difference between a state and a prop is that a prop can be passed from a parent component to the child component. This process is known as ‘prop drilling’.

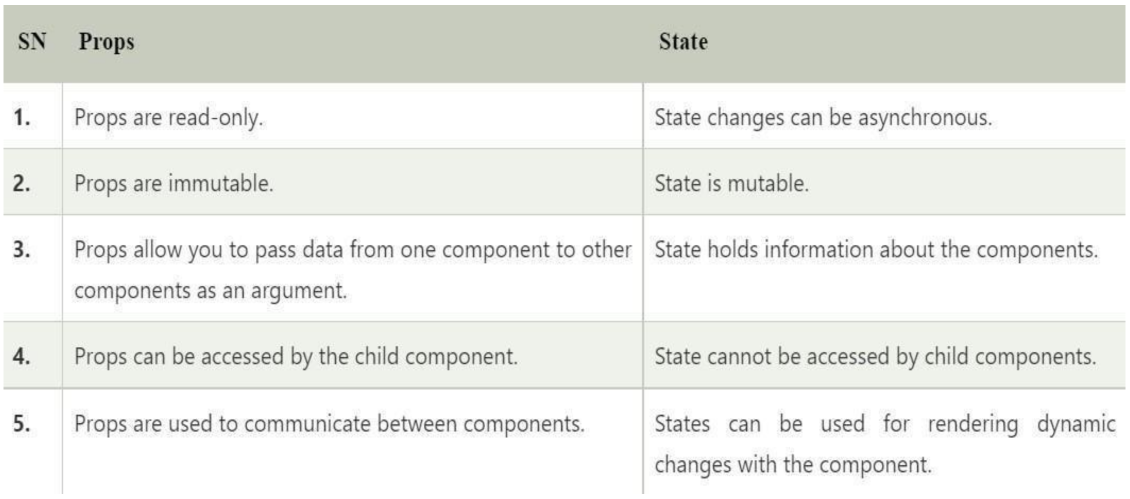
Eg:-

Adding an attribute called ‘brand’ to ‘Vehicle’ component :

```
const Ele = <Vehicle brand="Tata" />;
```

Passing the prop to the component :

```
function Truck(data) { return <h1>The price is :
  { data.price }</h1>;
}
```



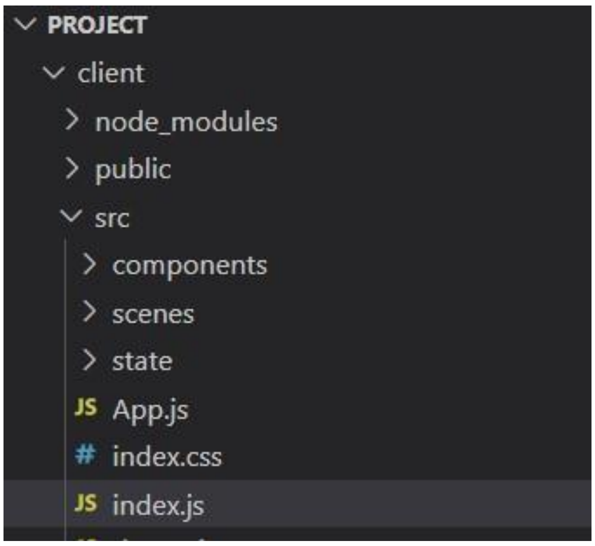
SN	Props	State
1.	Props are read-only.	State changes can be asynchronous.
2.	Props are immutable.	State is mutable.
3.	Props allow you to pass data from one component to other components as an argument.	State holds information about the components.
4.	Props can be accessed by the child component.	State cannot be accessed by child components.
5.	Props are used to communicate between components.	States can be used for rendering dynamic changes with the component.

Table 3.3 : Props vs State in react

3.4 Connecting Front-end and Back-end through APIs

Workspace of a typical react application is divided in two folders : Client and Server. This way of developing an application makes the development process clean, easy and the code is more readable. This is called ‘separation of concerns’. Client side directory’s main concern is user interface while the server side handles database, authentication and routing.

To connect these two directories, we use APIs. This file is present in the client side inside the ‘src’ folder (shown in Fig 3.19):



```
PROJECT
  client
    node_modules
    public
    src
      components
      scenes
      state
      App.js
      index.css
      index.js
```

Fig 3.12 : Location of API file

Types of requests :

Get request : To get the data from the database.

Post request : To send data from frontend to the database.

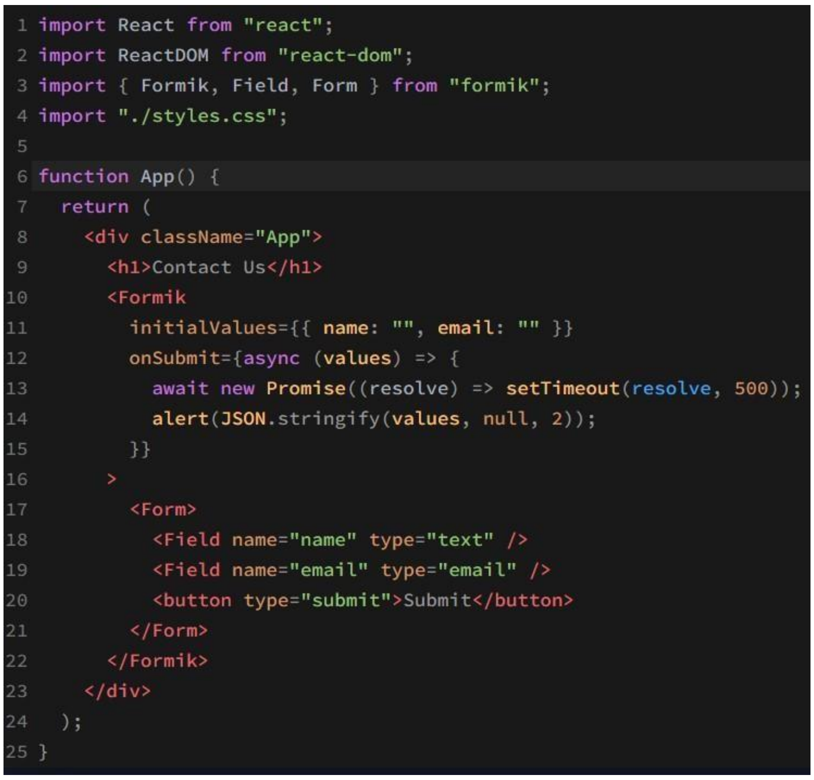
Patch request : To update an existing data / some part of an existing data. The request body only needs the part which needs to be updated.

Put request : Same functionality as Patch request. The only difference is that the body of put request needs to have the complete new data and not just the part which needs updating.

3.5 React Formik

Formik is a third-party library for React forms. Basic form programming and validation are offered. It is built on controlled components and drastically cuts down on form programming time. Let's use the Formik library to replicate the expenditure form as shown in Fig 3.20.

First, use the Create React App or Rollup bundler to create a new React application, such as React-formik-app, and then follow the instructions in the chapter on Creating a React application. cd /go/to/workspace npm install formik --save



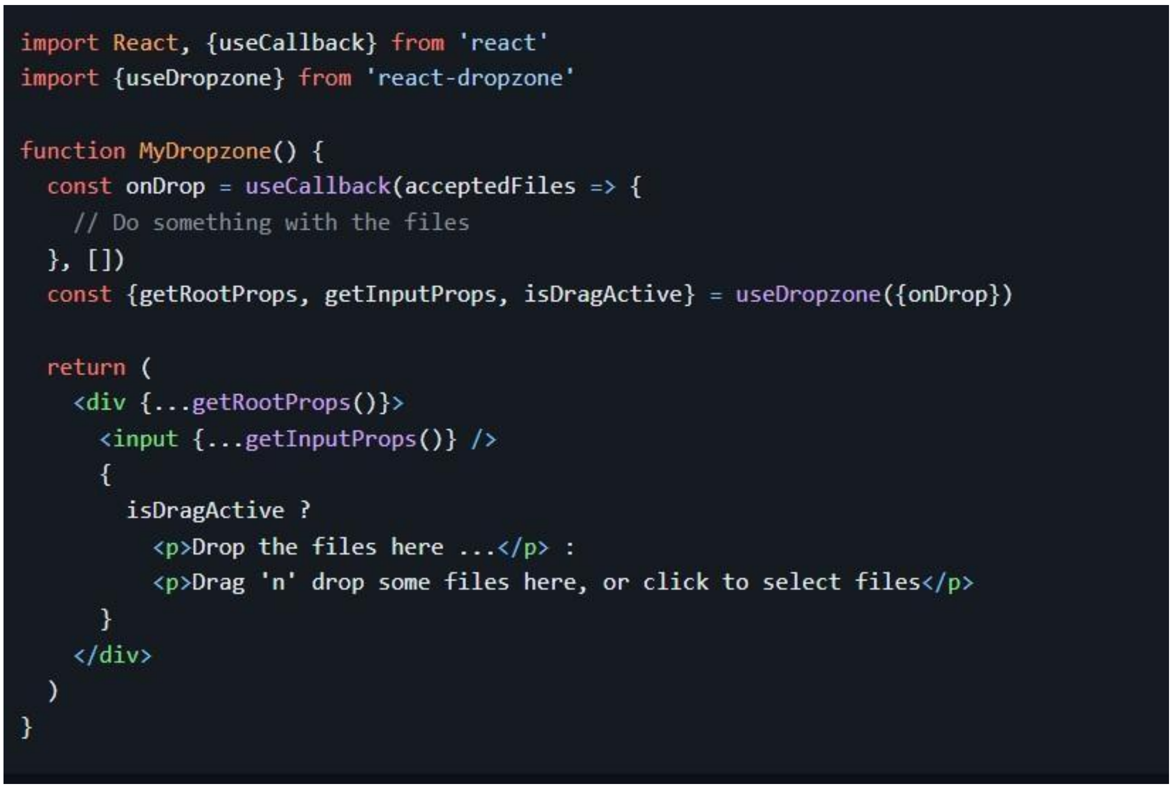
```
import React from "react";
import ReactDOM from "react-dom";
import { Formik, Field, Form } from "formik";
import "./styles.css";

function App() {
  return (
    <div className="App">
      <h1>Contact Us</h1>
      <Formik
        initialValues={{ name: "", email: "" }}
        onSubmit={async (values) => {
          await new Promise((resolve) => setTimeout(resolve, 500));
          alert(JSON.stringify(values, null, 2));
        }}
      >
        <Form>
          <Field name="name" type="text" />
          <Field name="email" type="email" />
          <button type="submit">Submit</button>
        </Form>
      </Formik>
    </div>
  );
}
```

Fig 3.13 Form creation using react formik

3.6 React Dropzone

npm install --save react-dropzone

Simply enough, the useDropzone hook binds the required handlers to establish a drag-and-drop zone. To obtain the props needed for drag and drop, use the getRootProps() function on any element. Use the getInputProps() fn and the returned props on an input> to get click and keydown behaviour.



```
import React, {useCallback} from 'react'
import {useDropzone} from 'react-dropzone'

function MyDropzone() {
  const onDrop = useCallback(acceptedFiles => {
    // Do something with the files
  }, [])
  const {getRootProps, getInputProps, isDragActive} = useDropzone({onDrop})

  return (
    <div {...getRootProps()}>
      <input {...getInputProps()} />
      {
        isDragActive ?
          <p>Drop the files here ...</p> :
          <p>Drag 'n' drop some files here, or click to select files</p>
      }
    </div>
  )
}
```

Fig 3.14 React dropzone using hooks

3.7 The Back-end implementation

Frameworks required : Expressjs and Mongoose

External Packages : dotenv and CORS

- Dotenv is a npm package for loading environment variables without manual programming.
- CORS or Cross-Origin Resource Sharing in Node. js is a mechanism by which a front-end client can make requests for resources to an external back-end server.
-

Backend setup

Initialising Express and setting up bodyparser :

const app = express(); dotenv.config();

app.use(bodyParser.json({limit: "30mb", extended: true}));

app.use(bodyParser.urlencoded({limit: "30mb", extended: true}));

Here we are initialising express and assigning it to a variable named ‘app’. Now this app variable can be used for routing.

Connecting to mongodb atlas :

const PORT = process.env.PORT || 3001; const

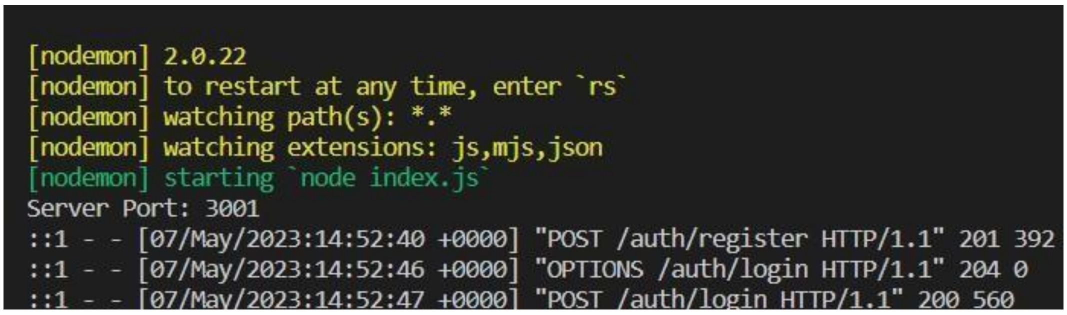
app = express();

mongoose.connect(process.env.CONNECTION_URL,{useNewUrlParser:true,useUnified Topology:true})

.then(()=> app.listen(PORT, ()=> console.log(`server running on ${PORT}`)))

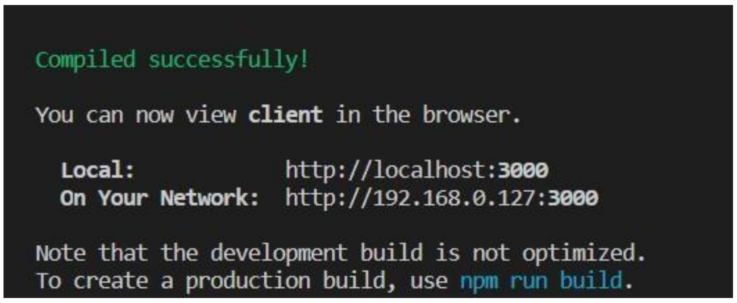
.catch((error)=>console.log(error.message));

Here we are setting our port as port 3001. This means that our client runs on localhost: 3000 while the server runs on localhost: 3001. After these set-ups, we can start our app by running ‘npm start’ command . The terminal is shown in Fig 3.22 and 3.23



```
[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Server Port: 3001
::1 - - [07/May/2023:14:52:40 +0000] "POST /auth/register HTTP/1.1" 201 392
::1 - - [07/May/2023:14:52:46 +0000] "OPTIONS /auth/login HTTP/1.1" 204 0
::1 - - [07/May/2023:14:52:47 +0000] "POST /auth/login HTTP/1.1" 200 560
```

Fig 3.15 : Server running on port 3001



```
Compiled successfully!

You can now view client in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.127:3000

Note that the development build is not optimized.
To create a production build, use npm run build.
```

Fig 3.16 : Client running on port 3000

3.8 MongoDB and Mongoose

Mongoose is an ODM library for MongoDB and Nodejs. It offers many different kinds of validation and also manages relationships between data. Some features of MongoDB are :

- Schema-less NoSQL
- Data stored in the form of json objects.
- No fixed structure.
- Fast as it is written in C++.
- Reduces complexity of deployment.

Terminologies

1. **Collections** : Multiple json documents together are called a collection. Collections are equivalent to tables in relational databases.
2. **Documents :** Documents can be compared to records / rows in a relational database. There is no concept of referencing data like SQL does in MongoDB. Mongo documents usually combine them in a document.
3. **Fields :** They are commonly known as properties or attributes. Fields are similar to columns in a table.
4. **Models :** Models are higher-order constructors that take a schema and create an instance of a document equivalent to records in a relational database.
5. **Schema Types :** SchemaTypes indicate the anticipated data type for specific fields, whereas Mongoose schemas define the overall form or shape of a document. Example : String, Number, Boolean.

MongoDB Atlas vs MongoDB Compass

Developers made MongoDB atlas so that people could scale and" deploy clusters with just a few clicks. The MongoDB team also developed and manages MongoDB Atlas, a global cloud database service. Enjoy the ease of use and automation of a fully managed service on your favourite cloud along with the flexibility and scalability of a document database. On the other hand, MongoDB Compass is described as "A GUI for MongoDB". Investigate your data visually. Ad hoc queries can be run quickly. Utilise all of the CRUD functionality to interact with your data. View and improve the performance of your queries.

Example :

```
const Schema = new mongoose.Schema({ name:
  {
    type: String, required:
    true
  },
  age: Number
});
const Sname = mongoose.model('Sch'', Schema);
```

In the code above, Sch defines the shape of the document which has two fields, name, and age. you can define the SchemaType for a field by using an object with a type property like the one used with the name field.

‘Name’ has two fields. Type denotes the data type while setting ‘required’ field to true makes it mandatory for the user to enter his name. This is not the same for the ‘age’ field. The data type for the ‘age’ field is a number but the absence of the ‘required’ field means that the user can continue without entering his age in the database / prompt.

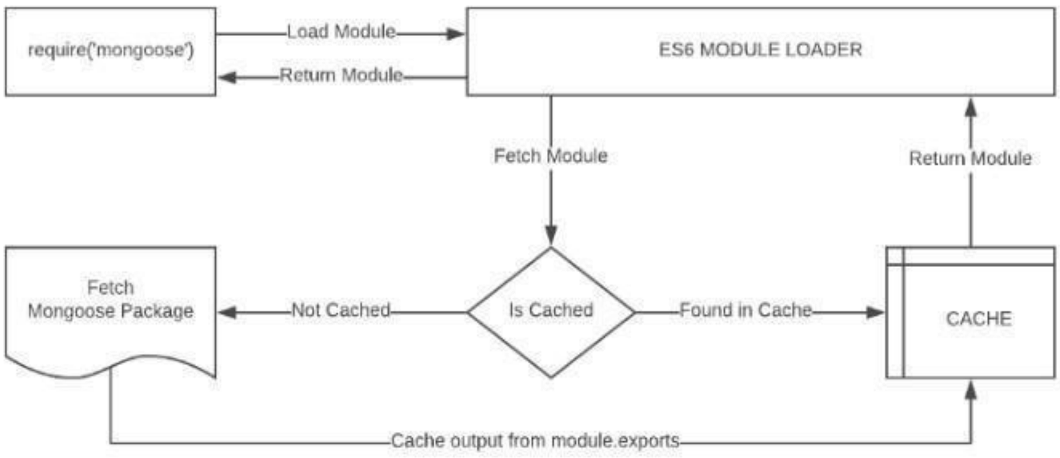


Fig 3.17 : Module import/require workflow

This project has two models : The user model and the post model.

User model : Made for letting the users Register and login. It has four fields : name, email, password and id. Name, email and password are the required fields while title is not mandatory.

Name is required as a user might create a post which requires his/her username to be displayed along with the post.

Email is required as the user may signin again after registering and we need to make sure that his id is saved in the database.

Password is required for validation.

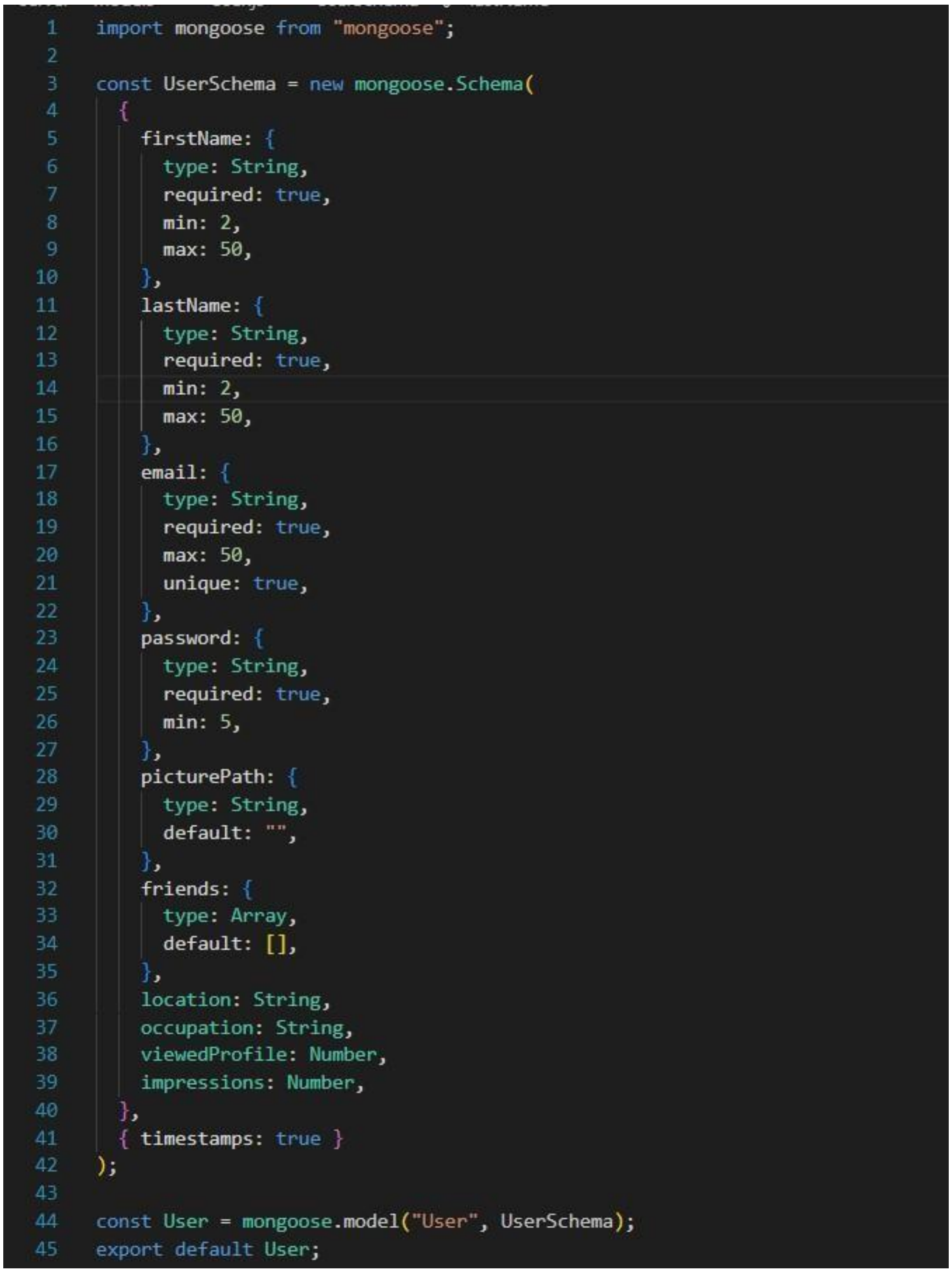
Title is not a required field as the user may not create a post in the session in which he logged-in.

3.8.1 Schemas

1. The User Schema

Holding a user collection where a user profile is saved and updated will be a basic strategy.

Afterward, each contact document may include owner information. These specifics vary depending on how the programme views the data and whether it needs to be provided or filtered. The basic minimum appears to be maintaining a userId for each contact or, if numerous users share the same contact information, perhaps an array of users. You can index and provide this userId to the contact collection query after logging in and identifying the user to retrieve only the necessary contacts as shown in Fig 3.25.



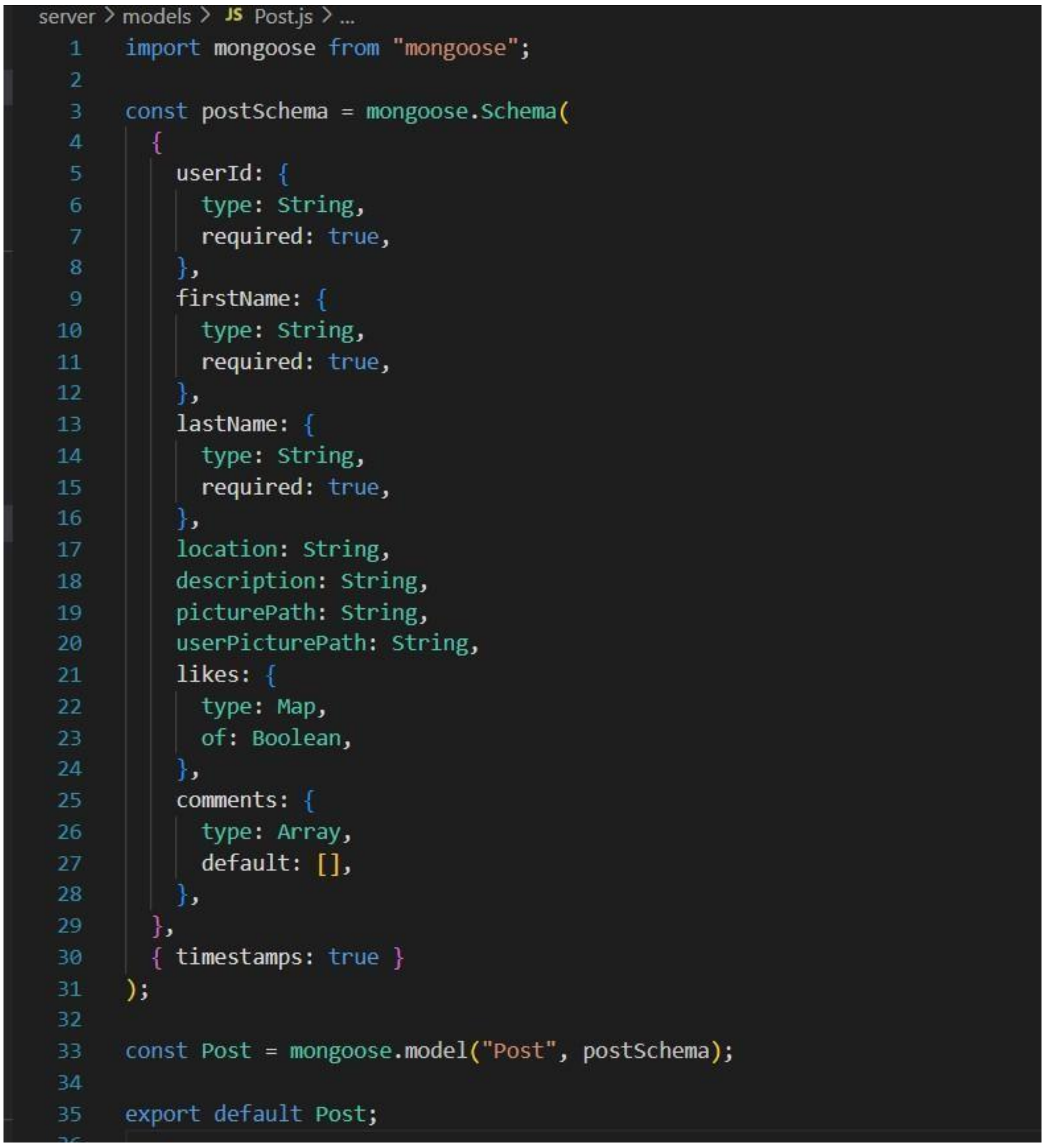
```
import mongoose from "mongoose";

const UserSchema = new mongoose.Schema(
  {
    firstName: {
      type: String,
      required: true,
      min: 2,
      max: 50,
    },
    lastName: {
      type: String,
      required: true,
      min: 2,
      max: 50,
    },
    email: {
      type: String,
      required: true,
      max: 50,
      unique: true,
    },
    password: {
      type: String,
      required: true,
      min: 5,
    },
    picturePath: {
      type: String,
      default: "",
    },
    friends: {
      type: Array,
      default: [],
    },
    location: String,
    occupation: String,
    viewedProfile: Number,
    impressions: Number,
  },
  { timestamps: true }
);

const User = mongoose.model("User", UserSchema);
export default User;
```

Fig 3.18 User Schema

2. The Post Schema

This schema is for the post a user might create. The fields present in this schema are title, name, message, createdAt, likes, tags.

None of the fields are required but likes and createdAt are 'default' fields which means they will always be present. The like field is an empty array by default while the date is set to the current date as shown in Fig 3.26.

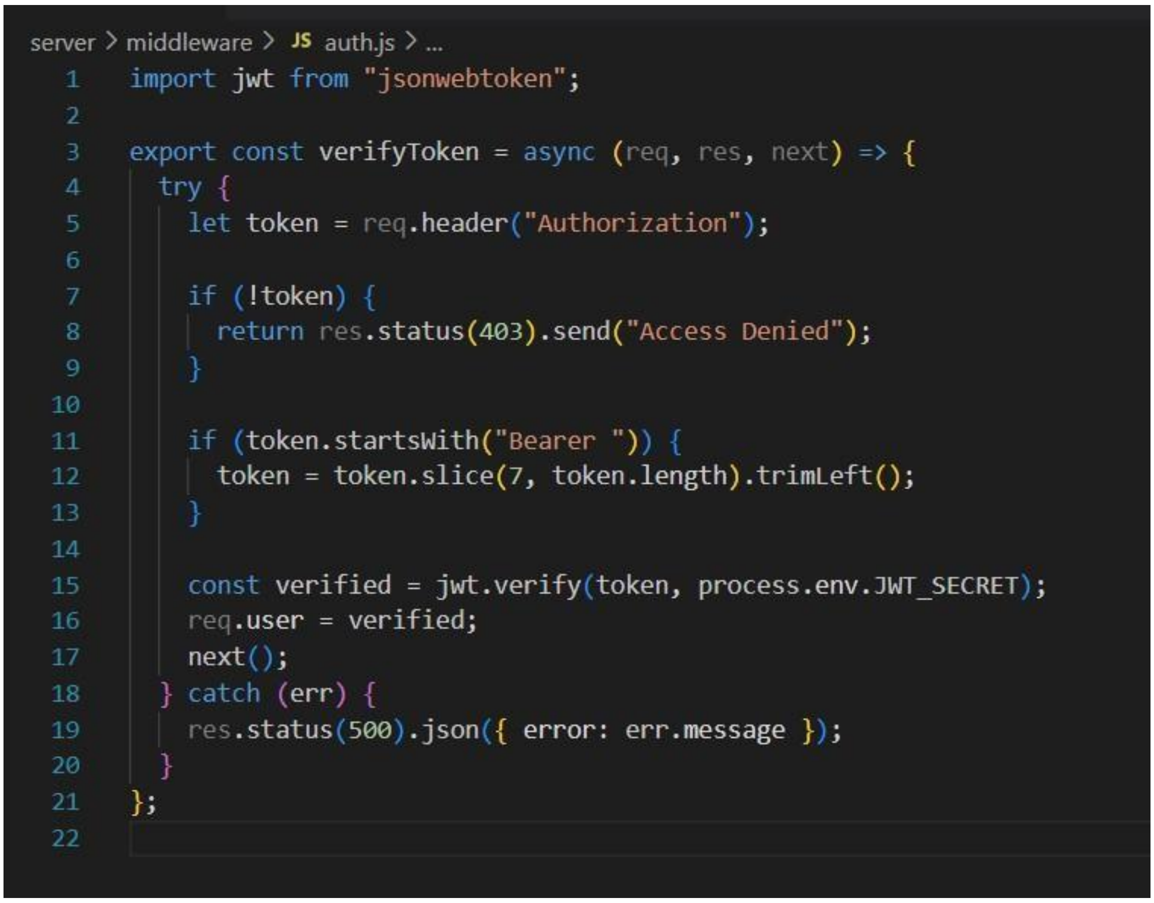


server > models > JS Post.js > ...

```
import mongoose from "mongoose";

const postSchema = mongoose.Schema(
  {
    userId: {
      type: String,
      required: true,
    },
    firstName: {
      type: String,
      required: true,
    },
    lastName: {
      type: String,
      required: true,
    },
    location: String,
    description: String,
    picturePath: String,
    userPicturePath: String,
    likes: {
      type: Map,
      of: Boolean,
    },
    comments: {
      type: Array,
      default: [],
    },
  },
  { timestamps: true }
);

const Post = mongoose.model("Post", postSchema);

export default Post;
```

Fig 3.19 : Post schemas

3.8.2 Middleware



```
server > middleware > JS auth.js > ...
import jwt from "jsonwebtoken";

export const verifyToken = async (req, res, next) => {
  try {
    let token = req.header("Authorization");

    if (!token) {
      return res.status(403).send("Access Denied");
    }

    if (token.startsWith("Bearer ")) {
      token = token.slice(7, token.length).trimLeft();
    }

    const verified = jwt.verify(token, process.env.JWT_SECRET);
    req.user = verified;
    next();
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};

```

Fig 3.20 : Middleware code for authentication

In the request-response cycle of an application, the middleware function in node.js is a function with complete access for making an object request, receiving an object response, and moving on to the next middleware function(as shown in Fig 3.27.

3.9 Controllers

The logic for sending data from the backend to be displayed in the frontend is handled by the controllers. The user requests to perform certain actions through API. The API sends the request to the backend where the controllers handle the logic to send data to the frontend. In this website we have two controller files. One to handle the requests related to a particular post and the other to handle the requests regarding security.

The Post Controller

It handles all the requests related to a particular post. The requests include :

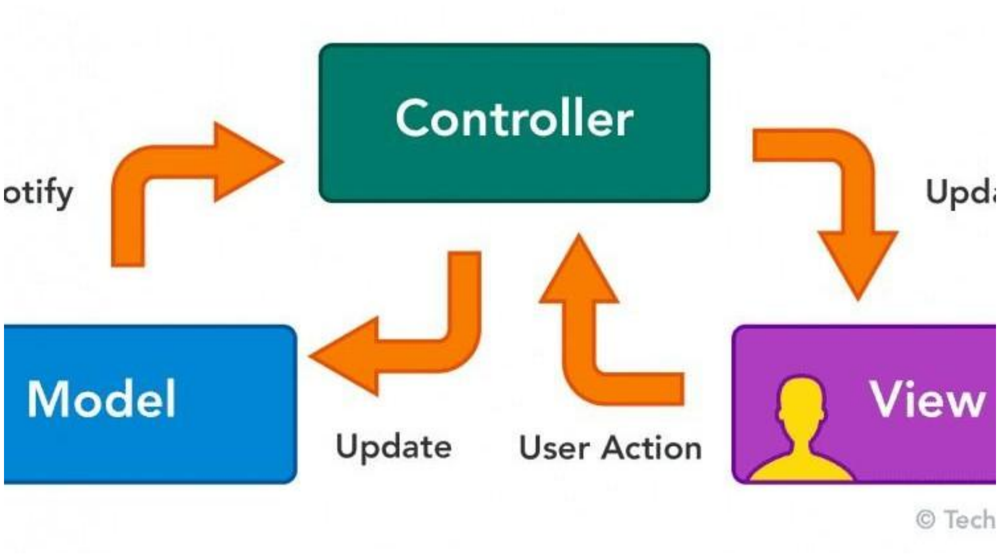
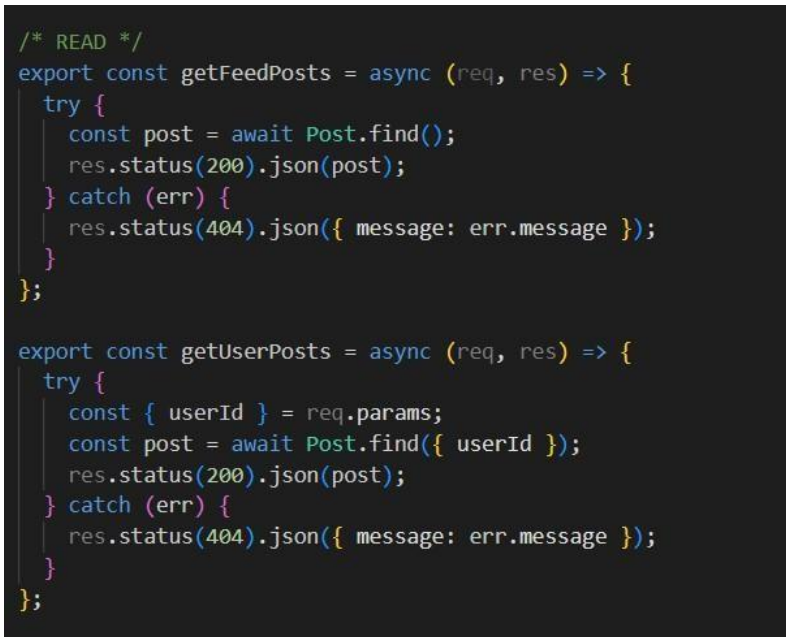


Fig 3.21 : Controllers

1. **Getting a post :**

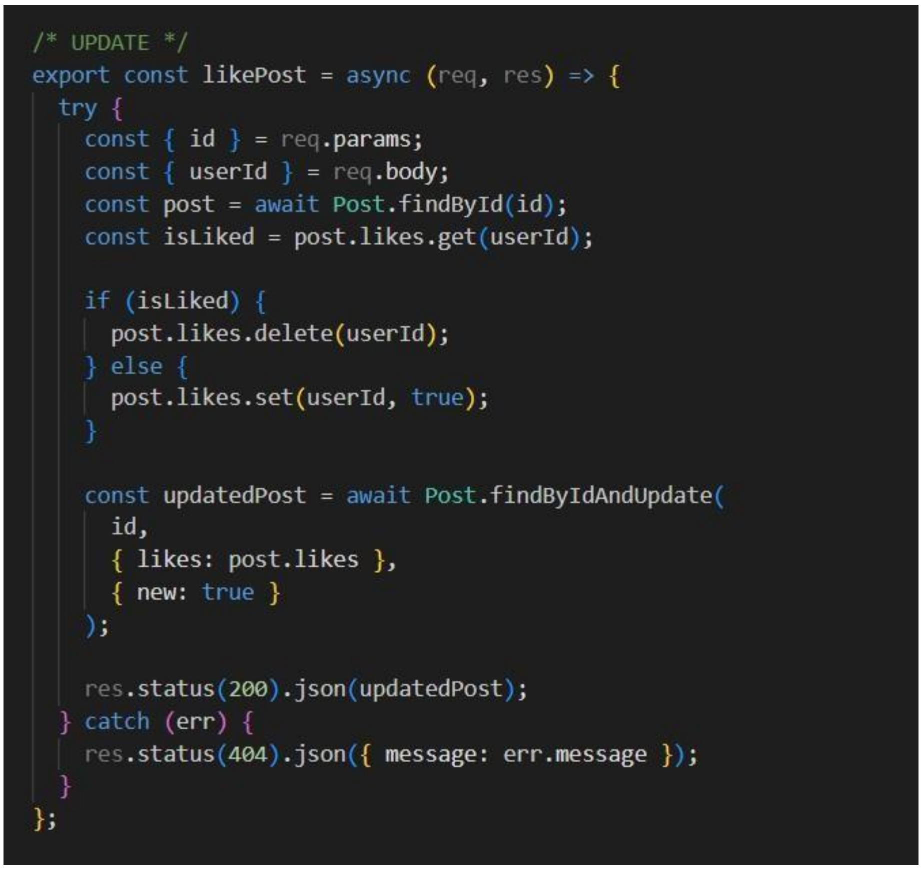
This function is called when the website is loading and all the available posts have to be displayed at once. The 'LIMIT' variable is set to 8 which means on a single page only 8 posts will be displayed. The remaining posts will be displayed on the next page. This is achieved through pagination.



```
/* READ */
export const getFeedPosts = async (req, res) => {
  try {
    const post = await Post.find();
    res.status(200).json(post);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};

export const getUserPosts = async (req, res) => {
  try {
    const { userId } = req.params;
    const post = await Post.find({ userId });
    res.status(200).json(post);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};
```

Fig 3.22 Getting a post **2.**

Updating a post

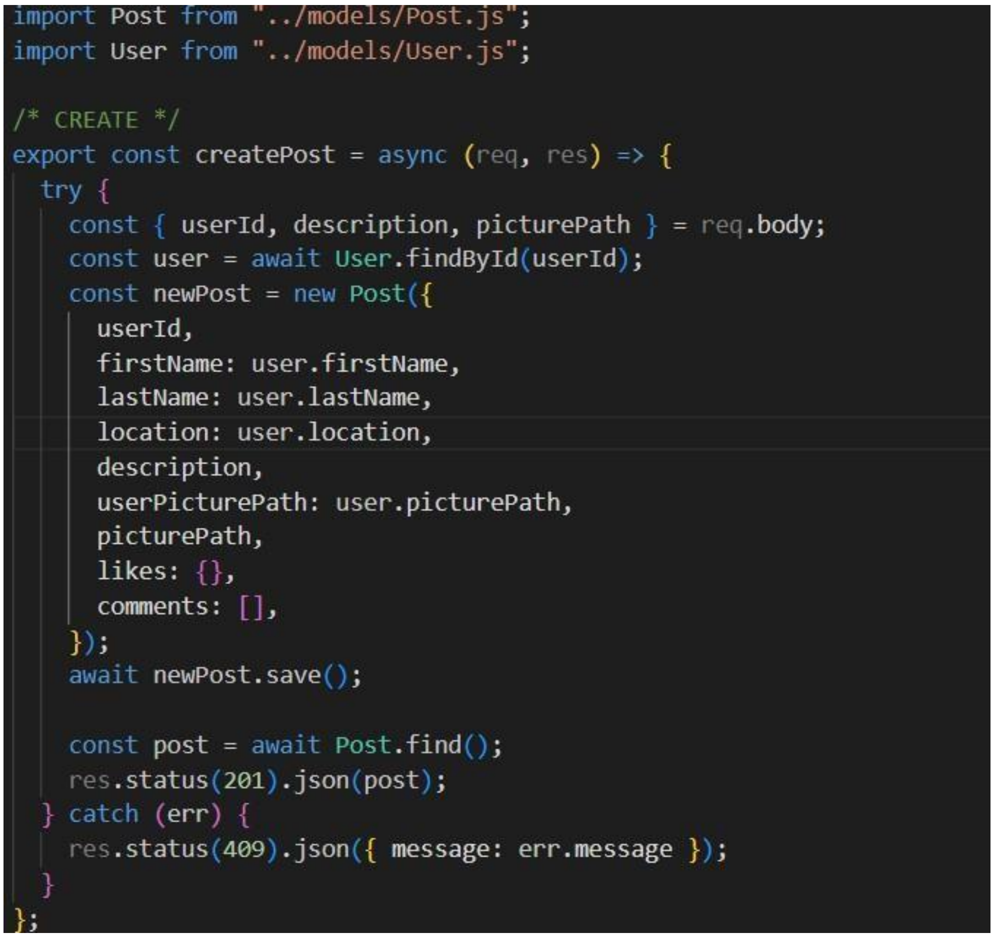


```
/* UPDATE */
export const likePost = async (req, res) => {
  try {
    const { id } = req.params;
    const { userId } = req.body;
    const post = await Post.findById(id);
    const isLiked = post.likes.get(userId);

    if (isLiked) {
      post.likes.delete(userId);
    } else {
      post.likes.set(userId, true);
    }

    const updatedPost = await Post.findByIdAndUpdate(
      id,
      { likes: post.likes },
      { new: true }
    );

    res.status(200).json(updatedPost);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};
```

Fig 3.23 Updating a Post

This is an asynchronous function which means it will take some time to execute but wont block other functions' execution. We are receiving '_id' from the database and storing it in the 'id' variable. If the status is 404, we are sending an error message otherwise we are updating the post by finding it by its id inside the database as shown in Fig 3.30.

3. Creating a post



```
import Post from "../models/Post.js";
import User from "../models/User.js";

/* CREATE */
export const createPost = async (req, res) => {
  try {
    const { userId, description, picturePath } = req.body;
    const user = await User.findById(userId);
    const newPost = new Post({
      userId,
      firstName: user.firstName,
      lastName: user.lastName,
      location: user.location,
      description,
      userPicturePath: user.picturePath,
      picturePath,
      likes: {},
      comments: [],
    });
    await newPost.save();

    const post = await Post.find();
    res.status(201).json(post);
  } catch (err) {
    res.status(409).json({ message: err.message });
  }
};
```

Fig 3.24 Creating a Post

This code defines an asynchronous function createPost that handles the creation of a new post. It expects a request (req) and a response (res) object as parameters.It extracts the userId, description, and picturePath from the request body. It retrieves the user information from the database using the User.findById(userId)method. It creates a new Post object with the extracted and retrieved data, including the user's name, location, and profile picture. The new post is saved to the database using newPost.save(). Finally, it fetches all posts from the database using Post.find() and sends the resulting posts as a JSON response with status 201 (Created). If any errors occur, it sends a JSON response with status 409 (Conflict) and an error message.

4. **Deleting a post** export const deletePost = async (req, res) => { const { id } = req.params; if(!mongoose.Types.ObjectId.isValid(id)) return res.status(404).send('No

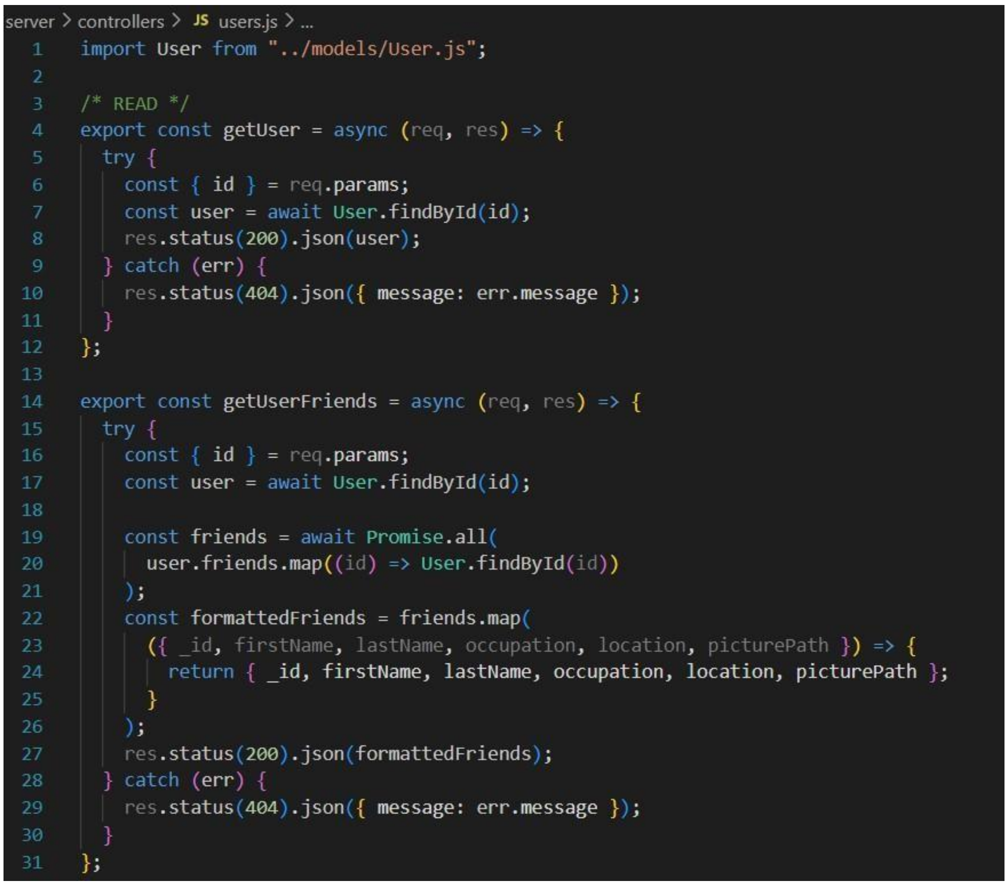
post found'); await PostMessage.findByIdAndRemove(id); res.json({ message: 'Post deleted successfully'});

}

The process is the same as updating the post. We are simply extracting the id from the request and then finding it inside the database and deleting it. The method used is ‘findbyIdAndRemove’ which is a standard function provided by mongoose.

The User Controller

1. Getting User and user friends



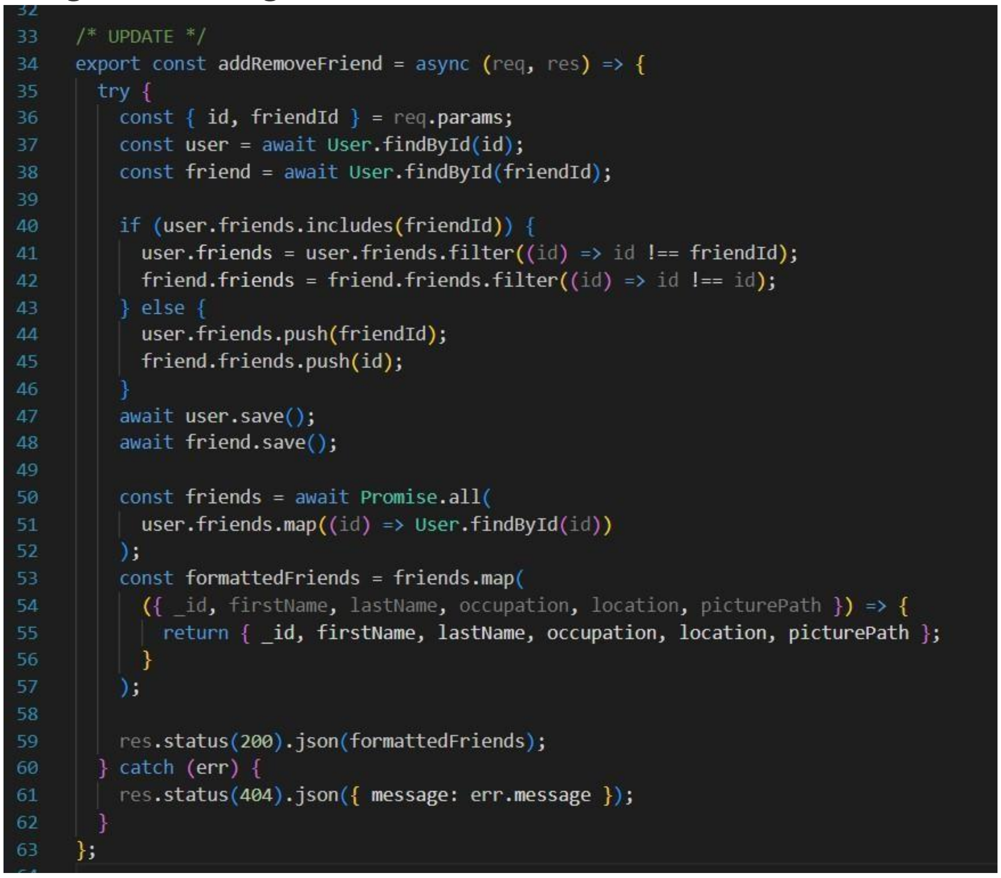
server > controllers > JS users.js > ...

```javascript
import User from "../models/User.js";

/* READ */
export const getUser = async (req, res) => {
  try {
    const { id } = req.params;
    const user = await User.findById(id);
    res.status(200).json(user);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};

export const getUserFriends = async (req, res) => {
  try {
    const { id } = req.params;
    const user = await User.findById(id);

    const friends = await Promise.all(
      user.friends.map((id) => User.findById(id))
    );
    const formattedFriends = friends.map(
      ({ _id, firstName, lastName, occupation, location, picturePath }) => {
        return { _id, firstName, lastName, occupation, location, picturePath };
      }
    );
    res.status(200).json(formattedFriends);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};
```

Fig 3.25 Getting User and user friends

Fig 3.32 shows that This code snippet exports two controller functions: getUser and getUserFriends. getUser retrieves a user's data by their id from the User model using User.findById(id). It then sends the user data as a JSON response with a status code of 200 (success) or 404 (not found) if an error occurs. getUserFriends follows a similar pattern as getUser. It retrieves the user's data and then fetches the data of their friends based on the user.friends array. The code uses Promise.all to perform parallel database queries to fetch each friend's data. The retrieved friend data is then formatted to include specific properties. Finally, the formatted friend data is sent as a JSON response with a status code of 200 or 404 if an error occurs. These controller functions assume the presence of a User model, which is imported from the "../models/User.js" file. The code demonstrates how to use the model's findById method to retrieve user data and manipulate the friend data before sending the responses.

2. Updating and removing user friends



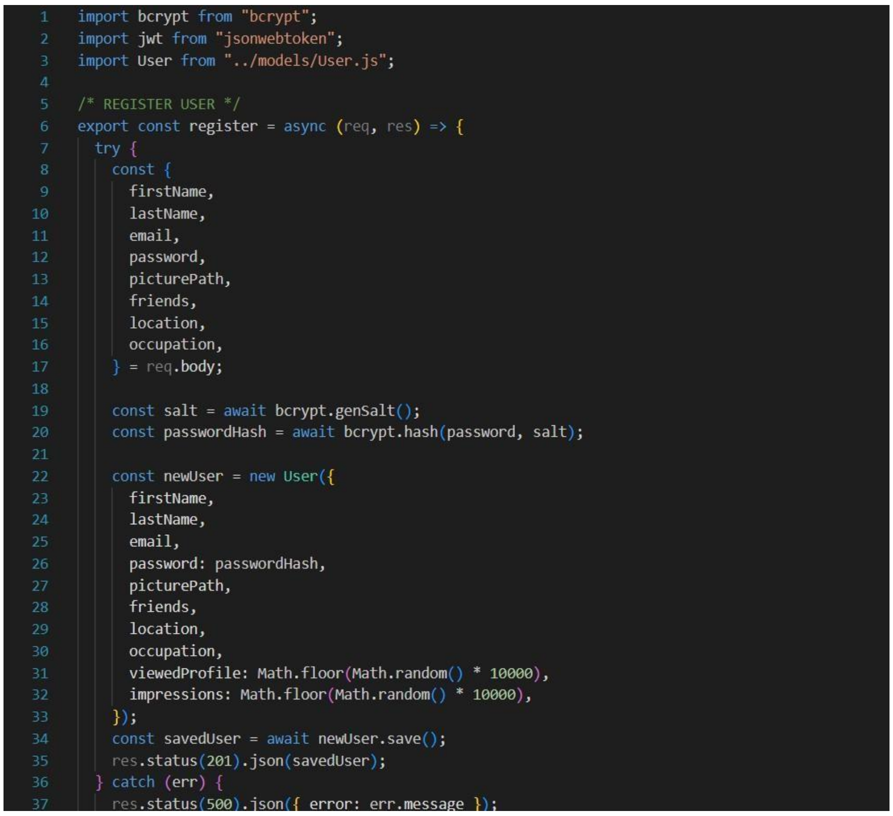
```javascript
/* UPDATE */
export const addRemoveFriend = async (req, res) => {
  try {
    const { id, friendId } = req.params;
    const user = await User.findById(id);
    const friend = await User.findById(friendId);

    if (user.friends.includes(friendId)) {
      user.friends = user.friends.filter((id) => id !== friendId);
      friend.friends = friend.friends.filter((id) => id !== id);
    } else {
      user.friends.push(friendId);
      friend.friends.push(id);
    }
    await user.save();
    await friend.save();

    const friends = await Promise.all(
      user.friends.map((id) => User.findById(id))
    );
    const formattedFriends = friends.map(
      ({ _id, firstName, lastName, occupation, location, picturePath }) => {
        return { _id, firstName, lastName, occupation, location, picturePath };
      }
    );

    res.status(200).json(formattedFriends);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};
```

Fig 3.26 Updating and removing user friends

This code is an example of an addRemoveFriend controller in JavaScript. It handles adding or removing friends for a given user based on the provided id and friendId parameters. The code performs the following steps: Retrieves the user and friend objects from the database using their respective IDs. Checks if the friend is already in the user's friend list. If so, it removes the friend from both the user's and friend's friend list arrays. Otherwise, it adds the friend to both arrays. Saves the updated user and friend objects back to the database. Retrieves the updated friend list for the user, including only the necessary fields. Sends a JSON response containing the formatted friend list or an error message if an error occurs.

The Authentication Controller

First we are finding a specific user by his email address. If the user doesn't exist, we send an error message along with 404 status. Then we check if the password entered by the user matches with the password saved in the database. If the password matches, we send a token for the user to remain signed in for some specific amount of time. If the credentials are wrong, we send an error message with the 400 status code. This shows in Fig 3.34.



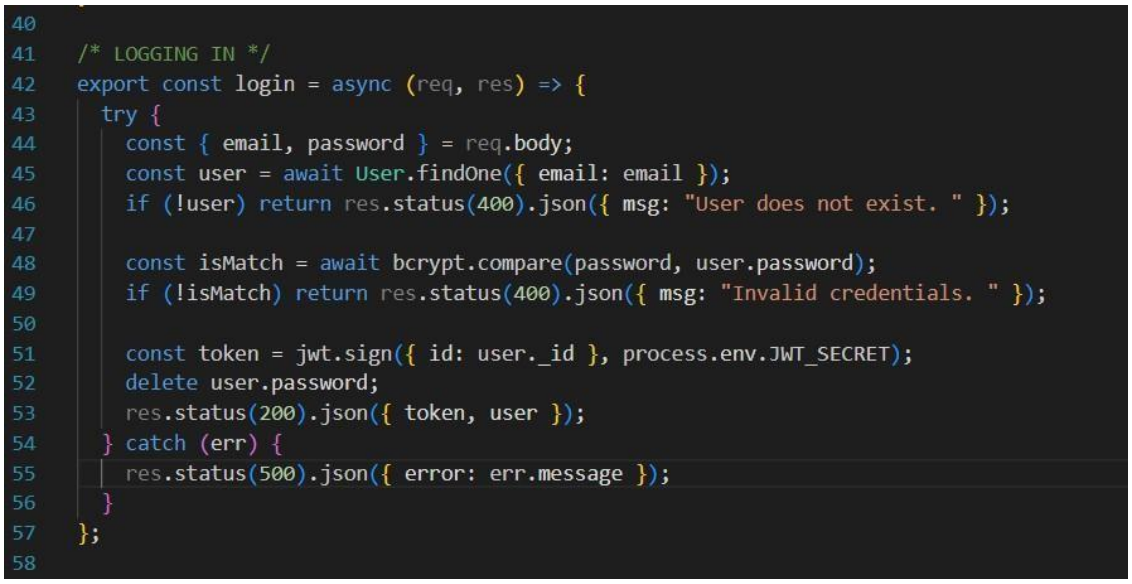
```javascript
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
import User from "../models/User.js";

/* REGISTER USER */
export const register = async (req, res) => {
  try {
    const {
      firstName,
      lastName,
      email,
      password,
      picturePath,
      friends,
      location,
      occupation,
    } = req.body;

    const salt = await bcrypt.genSalt();
    const passwordHash = await bcrypt.hash(password, salt);

    const newUser = new User({
      firstName,
      lastName,
      email,
      password: passwordHash,
      picturePath,
      friends,
      location,
      occupation,
      viewedProfile: Math.floor(Math.random() * 10000),
      impressions: Math.floor(Math.random() * 10000),
    });
    const savedUser = await newUser.save();
    res.status(201).json(savedUser);
  } catch (err) {
    res.status(500).json({ error: err.message });
```

Fig 3.27 Registering User

This code(Fig 3.35) is an implementation of a login functionality in a JavaScript controller. Here's an explanation in 5 lines The code receives a request (req) containing the user's email and password from the frontend. It searches for a user in the database (User.findOne({ email: email })) based on the provided email. If the user is not found, it returns a 400 status with a

JSON response indicating that the user does not exist. If the user is found, it compares the provided password with the stored password using bcrypt's compare method.

If the passwords match, a JSON Web Token (JWT) is generated (jwt.sign({ id: user._id }, process.env.JWT_SECRET)), and the token, along with the user object (with the password field removed), is returned as a JSON response with a 200 status. Otherwise, it returns a 400 status indicating invalid credentials or a 500 status in case of any server error.



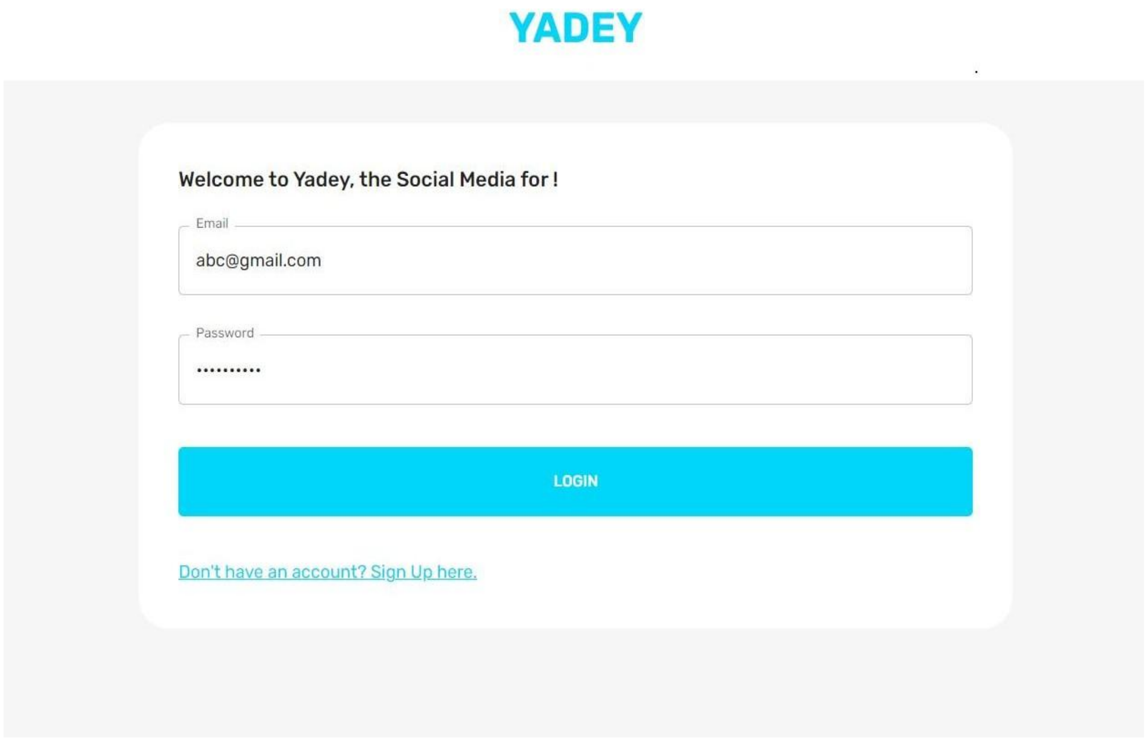
```
/* LOGGING IN */
export const login = async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email: email });
    if (!user) return res.status(400).json({ msg: "User does not exist. " });

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ msg: "Invalid credentials. " });

    const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET);
    delete user.password;
    res.status(200).json({ token, user });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

Fig 3.28 Logging user

Chapter 4

Result and Analysis

4.1 Login and Registration


YADEY

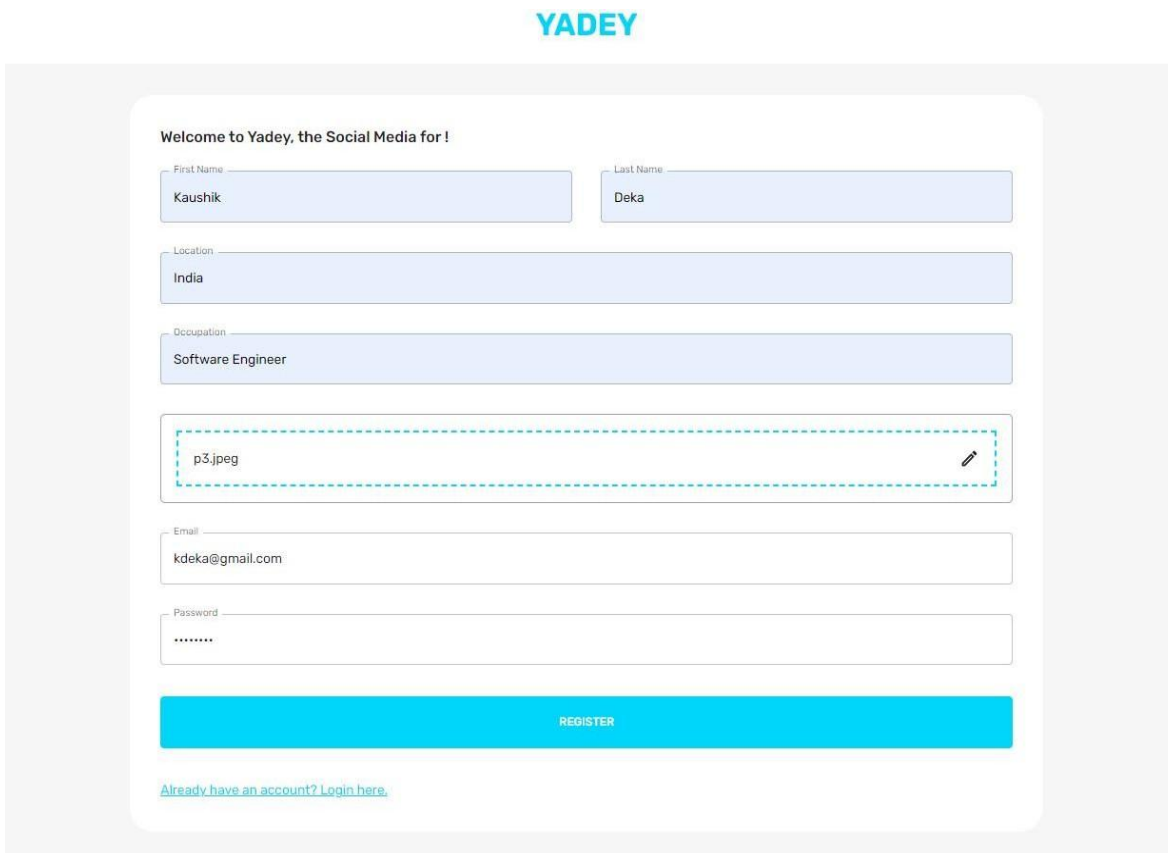
Welcome to Yadey, the Social Media for !

Email: abc@gmail.com

Password: ••••••••••

Fig 4.1 Log in page

If the user is not already registered in the server they can click on the "Don't have an account? Sign Up here." link and they will then be directed to the registration page. The user can then enter their details there along with the functionality to upload the profile picture.

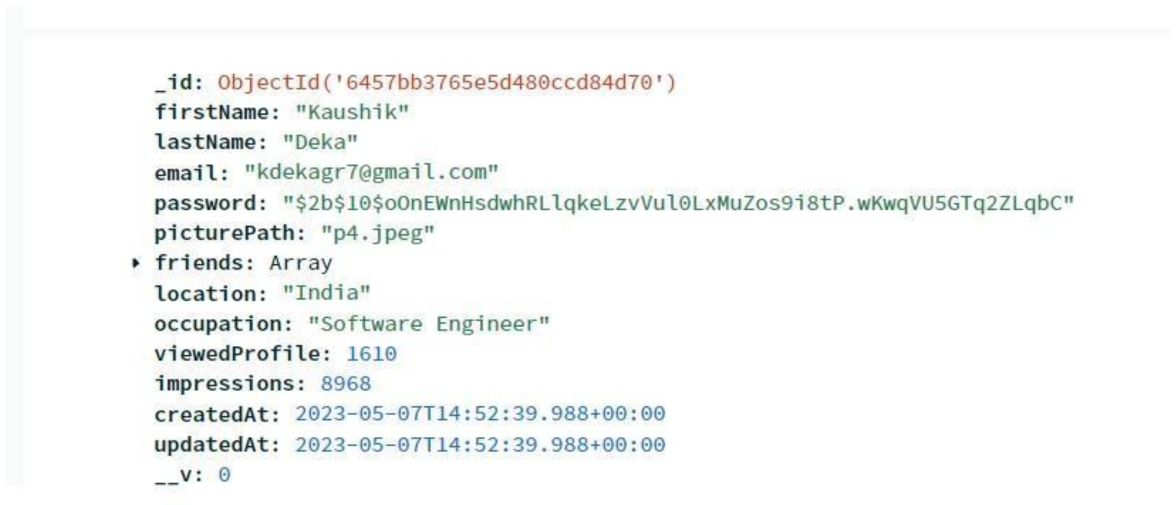

YADEY

Welcome to Yadey, the Social Media for !

First Name	Kaushik
Last Name	Deka
Location	India
Occupation	Software Engineer
	p3.jpeg
Email	kdeka@gmail.com
Password	········

Fig 4.2 Registration Page.

When the user has filled in all the information, they can click on the registration page which will send the data in JSON format along with the JWT authentication to the MongoDB running in the backend server. The user will then be redirected to the login page where they can login using the registered email address and password.



```
_id: ObjectId('6457bb3765e5d480ccd84d70')
firstName: "Kaushik"
lastName: "Deka"
email: "kdekagr7@gmail.com"
password: "$2b$10$oOnEWnHsdwhRLlqkeLzvVul0LxMuZos9i8tP.wKwqVU5GTq2ZLqbC"
picturePath: "p4.jpeg"
▸ friends: Array
location: "India"
occupation: "Software Engineer"
viewedProfile: 1610
impressions: 8968
createdAt: 2023-05-07T14:52:39.988+00:00
updatedAt: 2023-05-07T14:52:39.988+00:00
__v: 0
```

Fig 4.3 User info in the MongoDB database

So the user info they just entered is now registered in the database. Every user gets a unique hash id along with their password which is being encrypted by JWT.

4.2 Home Page

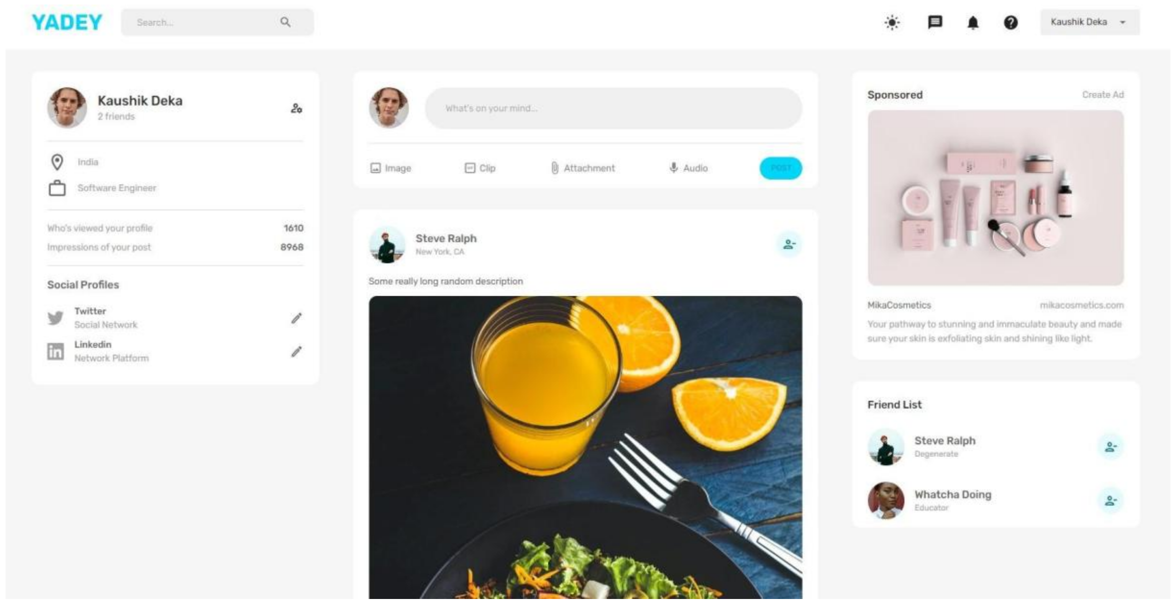


Fig 4.4 Clean looking homepage in Light Mode

The homepage contains all the information with all the widgets. The user can view the user information on the left side. Ads can be seen on the right side along with the friend list shown below. The user can write posts. They can see and like others' posts as well. They can also add friends using the add friend button. It is a clean looking UI with the option to change to dark mode as well using the button shown in Fig 4.5. The user can log out as well by clicking on the drop down on the navbar as shown in Fig 4.6. The dark mode UI is shown in Fig 4.7.

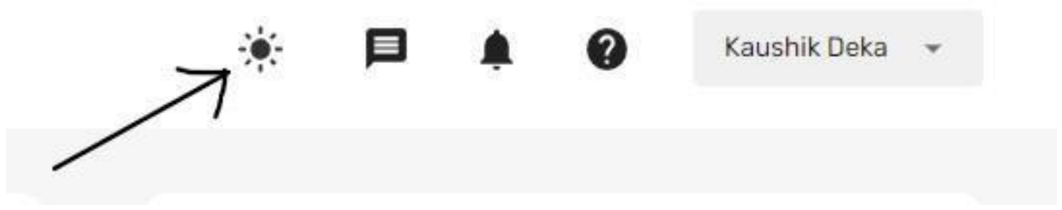


Fig 4.5 Button to change to Dark Mode.

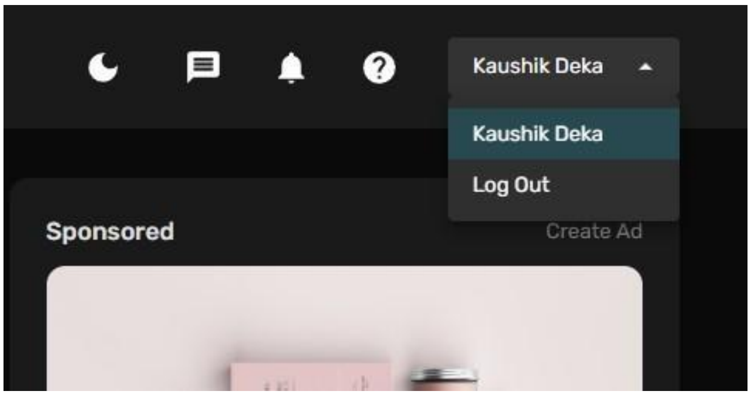


Fig 4.6 Logout option

There is the logout option as shown in Fig 4.6. This will deauthorize the users log in id from the backend server and change the value to null. logOut method takes no arguments and clears the currentUser data stored locally automatically. Create this function in the UserLogin component and call it in the onClick attribute of the logout button:

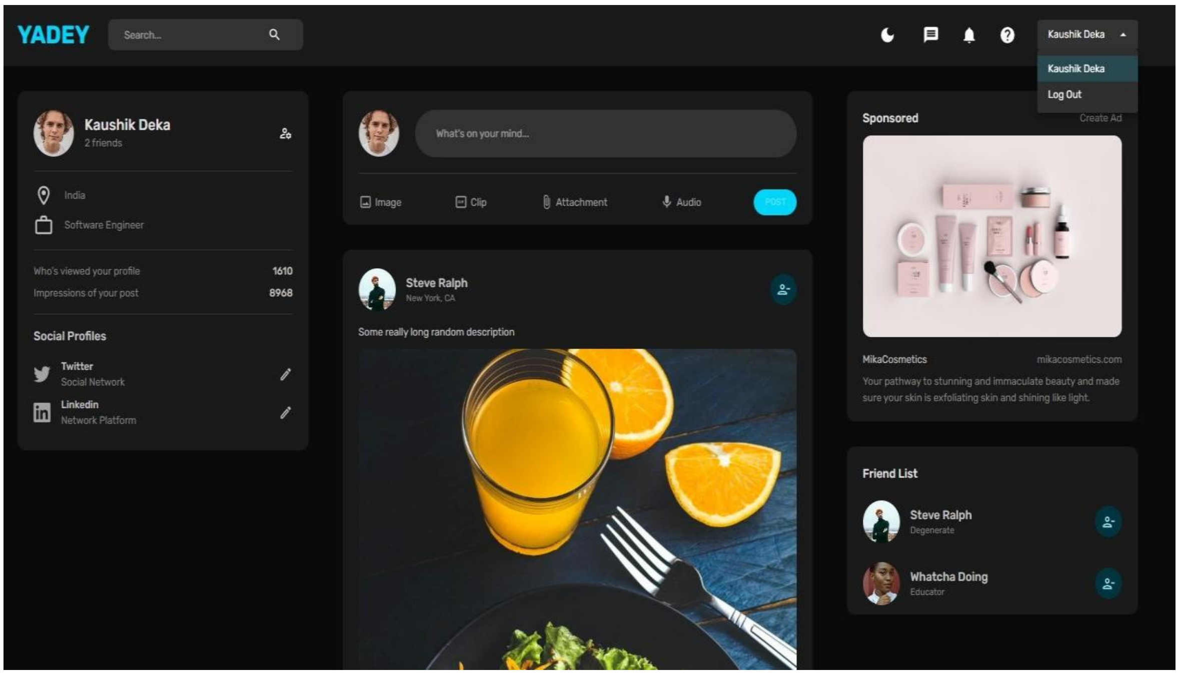
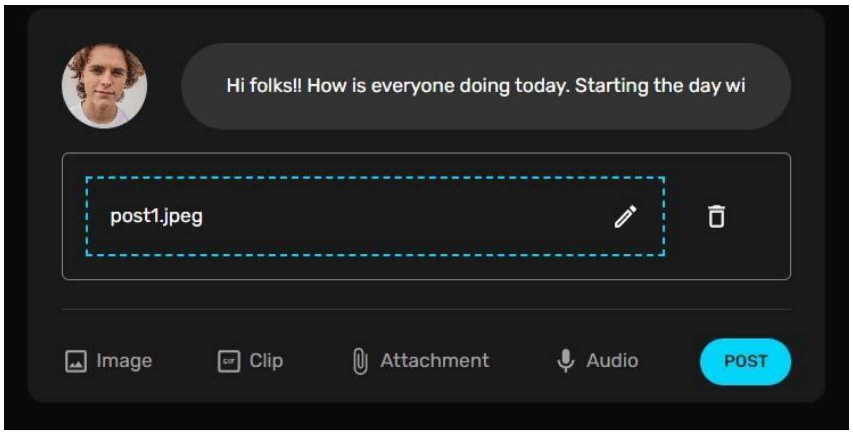


Fig 4.7 Homepage in Dark mode

The dark mode looks clean with contrasting colours to help viewers to distinguish the text and everything.It is neat little featured added as most of the modern applications nowadays has this feature

4.3 Creating and Searching a post

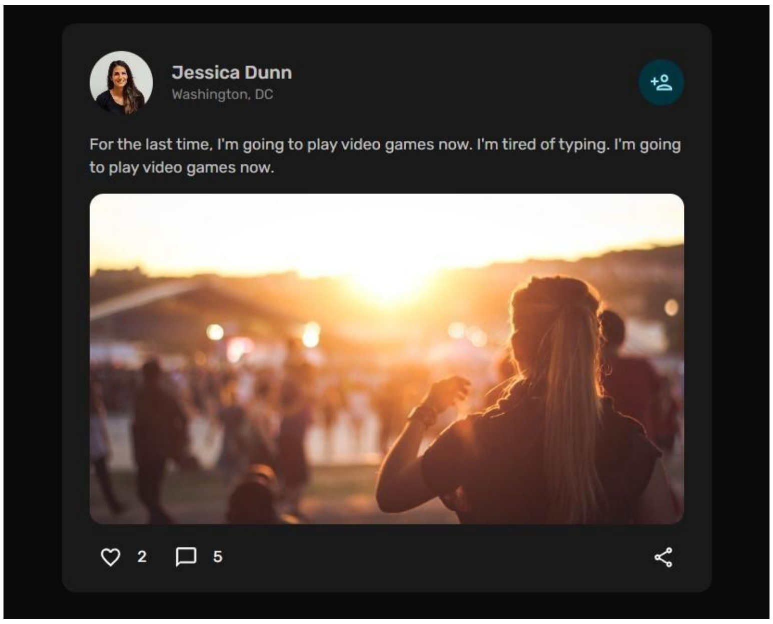
For creating a post, the user needs to enter many fields like title, tags, post message, image in jpg format. Along with these fields, the email id of the user and time of creation of the post is also entered in the database. The posts shown in Fig 4.8 and Fig 4.9 will be registered in the mongoDb database as shown in Fig 4.10.


Hi folks!! How is everyone doing today. Starting the day wi

post1.jpeg

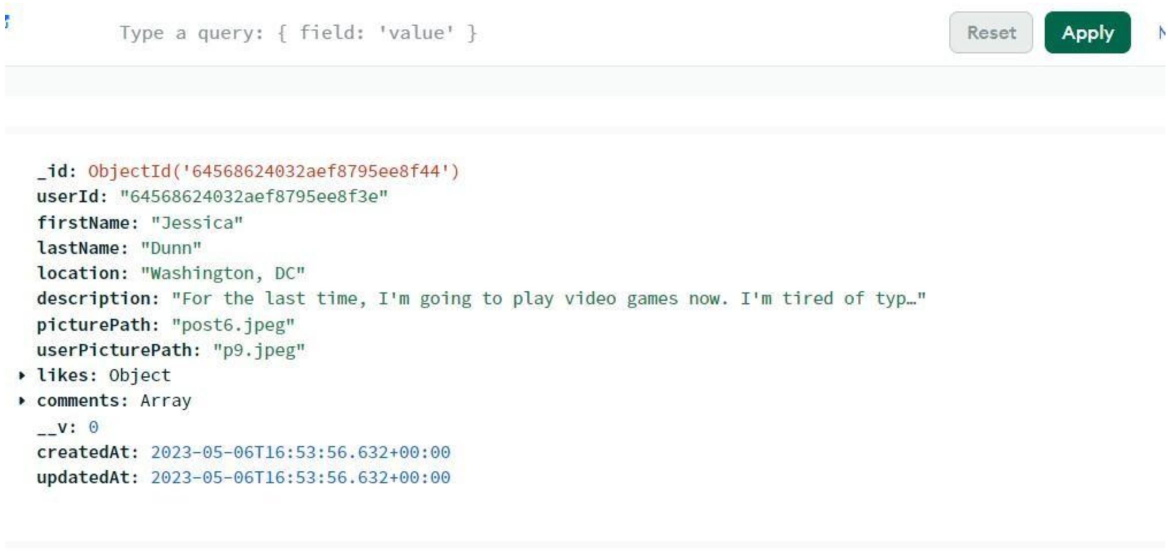
Fig 4.8 Post creation design

The Post button sends a request to the database to save the new post and the user credentials through the API. The clear button will clear all the fields at once.


Jessica Dunn
Washington, DC

For the last time, I'm going to play video games now. I'm tired of typing. I'm going to play video games now.

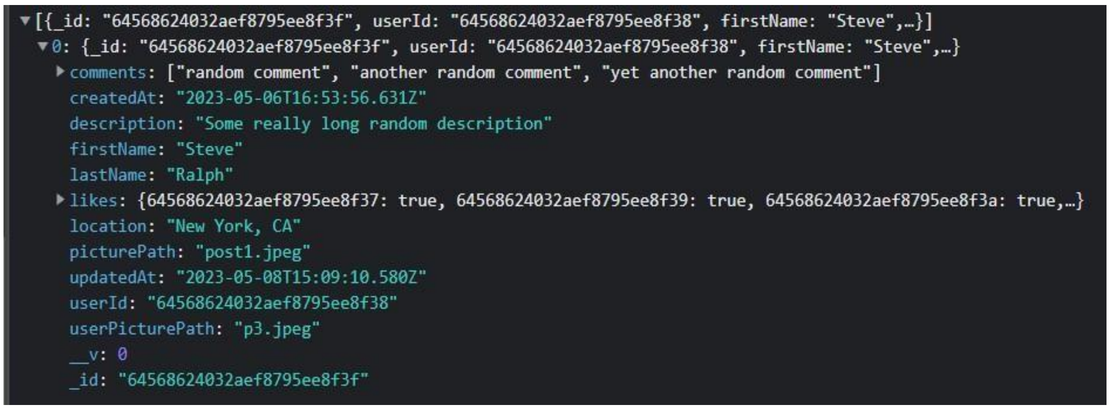
Fig 4.9 Post on homepage



```
_id: ObjectId('64568624032aef8795ee8f44')
userId: "64568624032aef8795ee8f3e"
firstName: "Jessica"
lastName: "Dunn"
location: "Washington, DC"
description: "For the last time, I'm going to play video games now. I'm tired of typ…"
picturePath: "post6.jpeg"
userPicturePath: "p9.jpeg"
▸ likes: Object
▸ comments: Array
__v: 0
createdAt: 2023-05-06T16:53:56.632+00:00
updatedAt: 2023-05-06T16:53:56.632+00:00
```

Fig 4.10 Post logged in the MongoDb database

We can see posts in the form of an array of objects with different fields like '_id', description, firstname, last name, location, image etc.



```
▾[{_id: "64568624032aef8795ee8f3f", userId: "64568624032aef8795ee8f38", firstName: "Steve",…}]
  ▾0: {_id: "64568624032aef8795ee8f3f", userId: "64568624032aef8795ee8f38", firstName: "Steve",…}
    ▸comments: ["random comment", "another random comment", "yet another random comment"]
     createdAt: "2023-05-06T16:53:56.631Z"
     description: "Some really long random description"
     firstName: "Steve"
     lastName: "Ralph"
    ▸likes: {64568624032aef8795ee8f37: true, 64568624032aef8795ee8f39: true, 64568624032aef8795ee8f3a: true,…}
     location: "New York, CA"
     picturePath: "post1.jpeg"
     updatedAt: "2023-05-08T15:09:10.580Z"
     userId: "64568624032aef8795ee8f38"
     userPicturePath: "p3.jpeg"
     __v: 0
     _id: "64568624032aef8795ee8f3f"
```

Fig 4.11 The post's metadata on the console

Above is the console view of the posts which will appear on the homepage once the website loads. We can see posts in the form of an array of objects with different fields like '_id', created at, description, picture path etc.

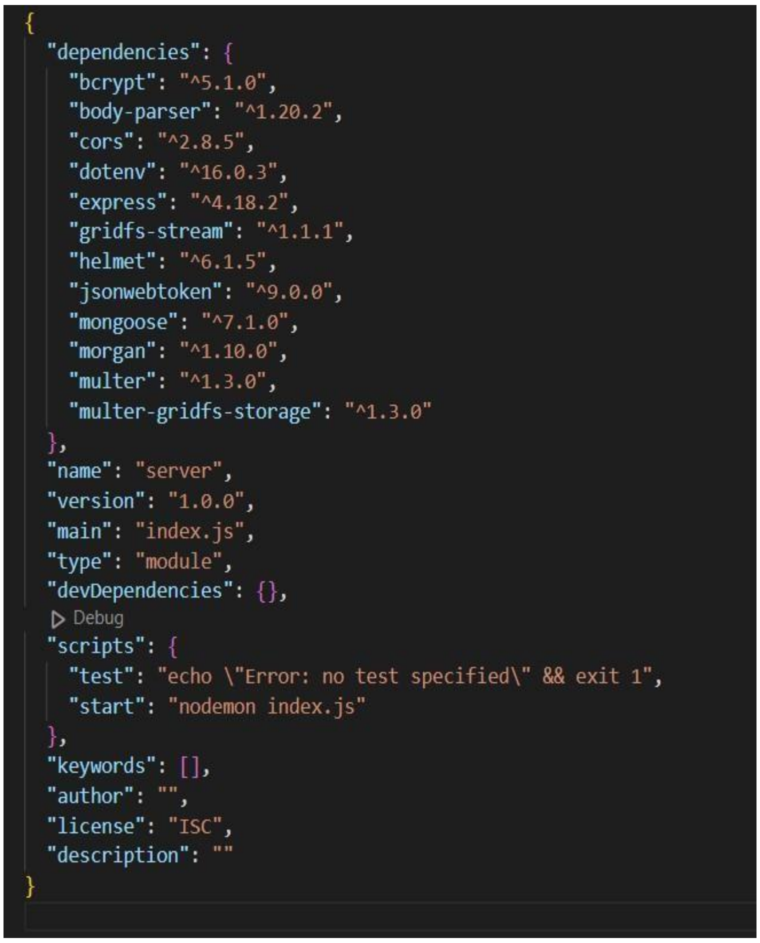
A particular post is a separate component called ‘Post’. Many of these components combine to make a component for all the posts of a single page called ‘Posts’. A single post has all the fields that the user enters manually like title, username, tags, message and the image. Apart from the manually entered fields, it has some automatically generated fields like the time of post creation and the email address of the user creating the post.

It also has a couple of buttons namely the like button and the delete button. The like button is available to all the logged in users while the delete button is available only to the creator of the post.

4.4 Performance analysis

Performed a **linter check** which makes sure that the program is properly formatted and follows standard code guidelines. There were **no** linter errors found in this project.

4.5 Final list of dependencies



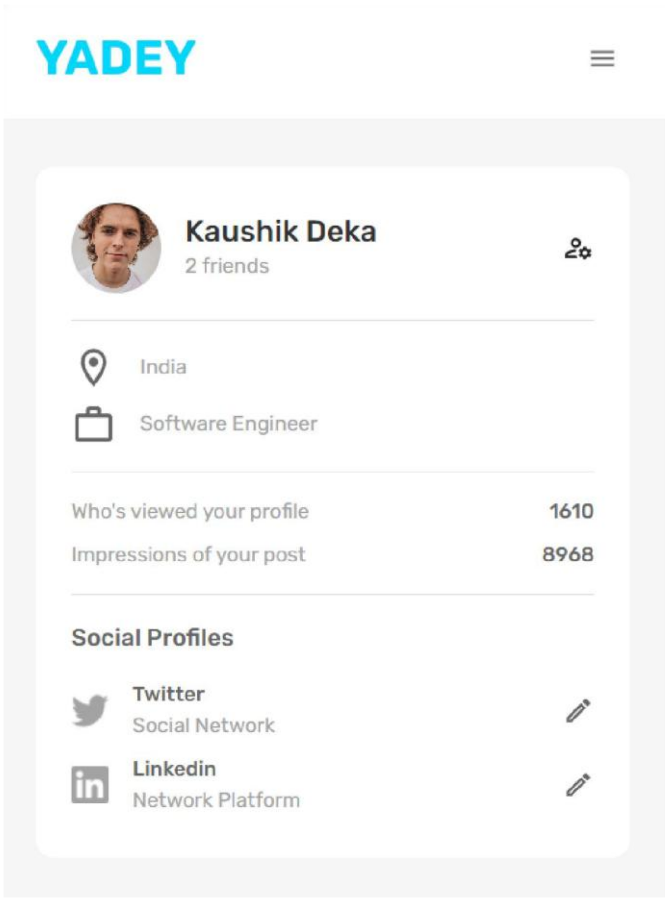
```json
{
  "dependencies": {
    "bcrypt": "^5.1.0",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "gridfs-stream": "^1.1.1",
    "helmet": "^6.1.5",
    "jsonwebtoken": "^9.0.0",
    "mongoose": "^7.1.0",
    "morgan": "^1.10.0",
    "multer": "^1.3.0",
    "multer-gridfs-storage": "^1.3.0"
  },
  "name": "server",
  "version": "1.0.0",
  "main": "index.js",
  "type": "module",
  "devDependencies": {},
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

Fig 4.12 : The package.json file

The package.json file stores the list of all the dependencies along with their version in json format which is a list of key-value pairs.

4.6 Mobile View

The entire social media application is completely responsive on any device. The view from an Iphone device is shown in Fig 4.12


YADEY

Kaushik Deka
2 friends

India
Software Engineer

Who's viewed your profile	1610
Impressions of your post	8968

Social Profiles

Twitter
Social Network

Linkedin
Network Platform

Fig 4.13 The app in mobile view

Chapter 5
CONCLUSION

5.1 Conclusions

The main aim of the training was to be able to understand and implement the concepts of Reactjs, MongoDB, ExpressJs and to be able to create a web application which could perform CRUD operations and can be tested using postman using the three layered architecture.

In this undergraduate project, the MERN stack and its features are examined and used to create a complete social media web application. The history of JavaScript, the foundation of the MERN stack, as well as the underlying theories, key ideas, and key methods of each individual technology have all been thoroughly covered in this essay. With the aid of a NoSQL database engine, the author has shown the benefits of those technologies and how they powerfully integrate to create an application with a connected backend and frontend. Following that, specific instructions for putting the social media application into practice were provided, demonstrating the viability of putting the aforementioned theories to use in solving a real-world issue. All things considered, the project's outcome can be deemed successful because all goals have been met. Given the project's time constraints, the application that was created satisfies all essential criteria for a social platform.

Through this project, I was able to achieve these goals. Doing this project has taught me that a developer should not only care about code but also keep in mind the users can make their experience better. The interface should be constant and smooth to allow the users to navigate through the website easier. Apart from that, writing clean and readable code is also important so that the other developers find it easier to understand and find bugs if there are any. We should try to write code which can be reused in future. Scope of enhancement is always there and we should try to learn from the feedback received.

5.2 Future Work

1. Through a graphical web user interface, users may interact with each other through chats while taking use of a secure login and authorisation approach.

2. Users can share their own own stories which can stay active for 24 hours
3. Adding the option of commenting on a particular post.
4. Adding ‘related posts’ section. This section will have all the posts related to the original post by location. To identify the location we will use either the title of the post or the tags provided by the user. Location refers to the country / continent.
5. Providing a separate and better ‘compose’ page.
6. The ability for users to add videos in posts.
7. The integration of NLP to group posts with similar hashtags.
8. The ability to integrate graph algorithms for friend recommendation systems similar to what Facebook has achieved.
9. The functionality of having an in app feature to take photos and videos with filters.
10. Providing third-party authentication.

References

1. Sourabh Mahadev Malewade, Archana Ekbote "Performance Optimization using MERN Stack on Web applications", publisher: IJRASET, vol. 10, doi : 6.06.2021, pp. 2278-0181.
2. Yogesh Baiskar, Priyas Paulzagade, Krutik Koradia, Pramod Ingole, Dhiraj Shirbhate "MERN : A Full stack development",publisher: IJRASET, vol. 1, doi : https://doi.org/10.22214/ijraset.2022.39982
3. Aarti Singh, Ananya Anikesh "Web development and Computer Science and Engineering", publisher : IJRASET, Vol 1, doi : https://doi.org/10.53555/cse.v2i4.612
4. Prakarsh Kaushik, Shashikant Suman, Basu Dev Shivahare, Vimal Bibhu "Web development and performance comparison of web development technologies in Nodejs and python", publisher : ICTAI, doi : 10.1109/ictai53825.2021.9673464
5. Pratiksha D Dutonde "Website development technologies : A review", publisher: IJRASET, vol. 10(1), doi : 10.22214/ijraset.2022.39839, pp. 359-366.
6. Stonebraker, Michael. "SQL databases v. NoSQL databases." Communications of the ACM 53.4 (2010): 10-11.
7. Aboutorabi[a], S.H., Rezapour, M., Moradi, M. and Ghadiri, N., 2015, August. Performance evaluation of SQL and MongoDB databases for big e-commerce data. In 2015 International Symposium on Computer Science and Software Engineering (CSSE) (pp. 1-7). IEEE.
8. Chodorow, C. "Introduction to mongoDB." Free and Open Source Software Developers European Meeting (FOSDEM). 2010. [18] Tilkov, Stefan, and Steve Vinoski. "Node. js: Using JavaScript to build high-performance network programs." IEEE Internet Computing 14.6 (2010): 80-83.
9. Boicea, A., Radulescu, F., Agapin, L. I. (2012, September). MongoDB vs Oracle--database comparison. In 2012 third international conference on emerging intelligent data and web technologies (pp. 330-335). IEEE.