Social Media Web Application

Project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering/Information Technology

By

Shubham Singh (191402)

Under the supervision of

Dr. Jagpreet Sidhu

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh

Certificate

We hereby declare that the work presented in this report entitled "Social media web application" in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering/Information Technology submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from January 2023 to May 2023 under the supervision of **Dr. Jagpreet Sidhu** (Assistant Professor (SG) CSE & IT, JUIT).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature)

Shubham Singh (191402)

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature)

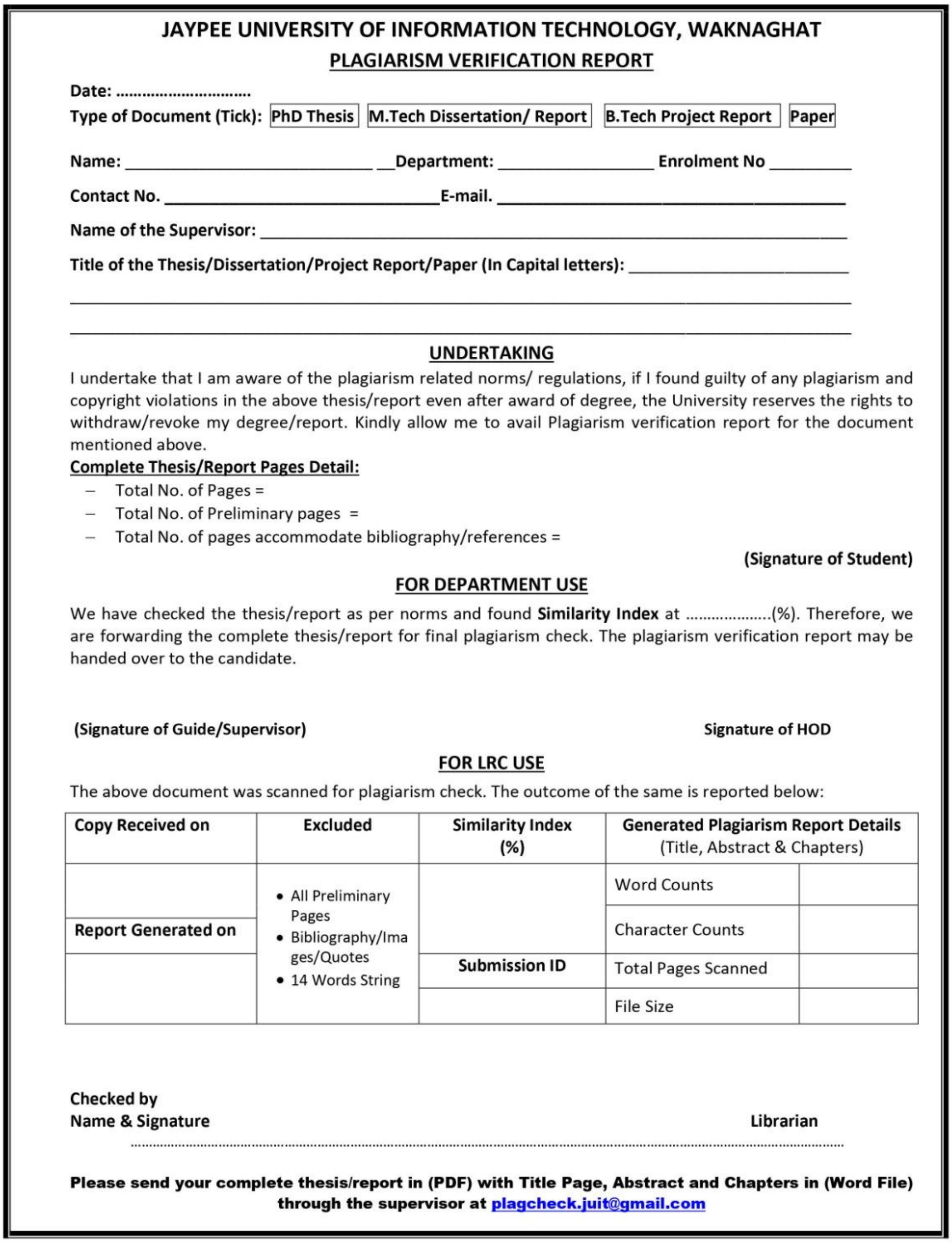
Supervisor Name

Designation

Department name
Dated:

Plagiarism Certificate

As provided by the LRC of JUIT.



JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ....

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: ___________________________ __**Department:** _________________ **Enrolment No** _________

Contact No. ______________________________**E-mail.** ______________________________________

Name of the Supervisor: ____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): ________________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

– Total No. of Pages =
– Total No. of Preliminary pages =
– Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	**Excluded**	**Similarity Index (%)**	**Generated Plagiarism Report Details** (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature **Librarian**

....

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

Acknowledgement

We are thankful to all the people who joined as part of making this journey of fulfilling this project into a working model. We are grateful to Jaypee University of Information technology for giving us a wonderful platform for exploring our software developing skills during the making of this project.

Additionally, we would like to extend our sincere appreciation to Dr. Vivek Sehgal, Head of the Department, for providing all the assistance and support necessary for the project's conception, execution, and presentation.

We are also thankful to our mentor Dr Jagpreet Sidhu as well as other staff members of the Computer Science and Engineering department, Jaypee University of Information Technology for their constructive and helpful inputs.

Table of Contents

List of Abbreviations

ABBREVIATIONS	**FULL-FORM**
API	Application Programming Interface
NPM	Node Package Manager
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
HTML	Hyper text Markup Language
JSX	JavaScript XML
SQL	Structured Query Language
URL	Uniform Resource Locator
UI	User Interface

List of Figures

List of Tables

Abstract

The goal of the website is to allow the users to post their experience and photos of the places they travelled to. This project was assigned to me by the company in order to help me get a better understanding of the skills that are needed in the area of website design and development. This website was supposed to be the implementation of the knowledge that we have acquired while learning web designing and development in the last four to five months. This website follows a three-tier architecture. The presentation tier is kept simple and user friendly. The main purpose of this tier is to display information and to collect information from the user. The application tier is the heart of web applications. All the information about the user collected from the presentation tier is stored and processed here. It is mainly implemented using nodejs. EJS is the templating engine used. The last tier is the data tier where information processed by the application tier is stored. Information about users such as their login ids, passwords(hashed) , likes, comments and feedback all are stored using this layer. In this website, the data tier is implemented using Mongodb(NoSQL). Mongoose is the library used.

The three-tier architecture ensured faster development as each tier can be developed simultaneously. It also ensures improved scalability, reliability and security

1. INTRODUCTION

1.1 Introduction

It is a basic web application that implements CRUD operations based on the three layered architecture. Programs at each layer have their own unit test. 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 expressjs, passport,js (for authentication and security). Its library Mongoose is used to write more readable code. For authentication , we are using Passport.js.
The Website allows users to login/signup, read a post, search a particular post. Creation, deletion and liking a post is only allowed if the user is logged-in.

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

The front-end of this website is developed exclusively using Reactjs. Material-ui is used for styling instead of plain CSS. This combination allows faster development and scalability. Each component can be developed simultaneously and error in one component won't affect other components. For the backend we have used expressjs for making APIs and MongoDB as database. Mongoose is a framework for MongoDB and Expressjs.

Executing HTTP requests is React's responsibility. They can set up dynamic data downloads in this way without having to reload the website. This makes the website significantly faster than usual.

1.5 Organisation

Dedicated to creating Commercially Scalable Blockchain Products, **Nonceblox** is a team of gifted blockchain architects, consultants, business SMEs, and cryptocurrency advisors. Solutions on Hyperledger, Polygon, Solana, BSC, and Polkadot are developed by Nonceblox. Our talented team of blockchain designers, specialists, industry SMEs, and cryptocurrency consultants have a passion to develop blockchain technologies that are useful to businesses and are economically feasible.

NonceBlox

Fig 1: Company logo

2. LITERATURE SURVEY

1. React documentation

React is a front-end framework that allows users to write reusable code for each component of a website. Each component can be combined to develop a complete user interface.

2. Javascript documentation - Mozilla

Writing basic to advanced level asynchronous javascript for developing a fully-fledged and scalable web application.

3. Git and Github

Official documentary that familiarises you with the concepts of a version control system i.e Git and how it works with GitHub.

4. Performance optimization using mern stack

The author of "Performance optimization using mern stack" Sourabh Mahadev Malwade talks about ways of improving performance of web applications developed using mern stack.

5. MERN: A Full-Stack development

A journal by Yogesh Baiskar and published by IJRASET talks about the methodology and chronology of full stack development. This document is covering the journey of a full-stack development from a frontend to deploying a site.

6. MongoDB documentation

MongoDB uses it's library - mongoose to make the code more readable, concise and efficient. Mongoose is used widely to enhance the code readability.

7. Introducing JSX - React

3. SYSTEM DEVELOPMENT

3.1 System design diagram

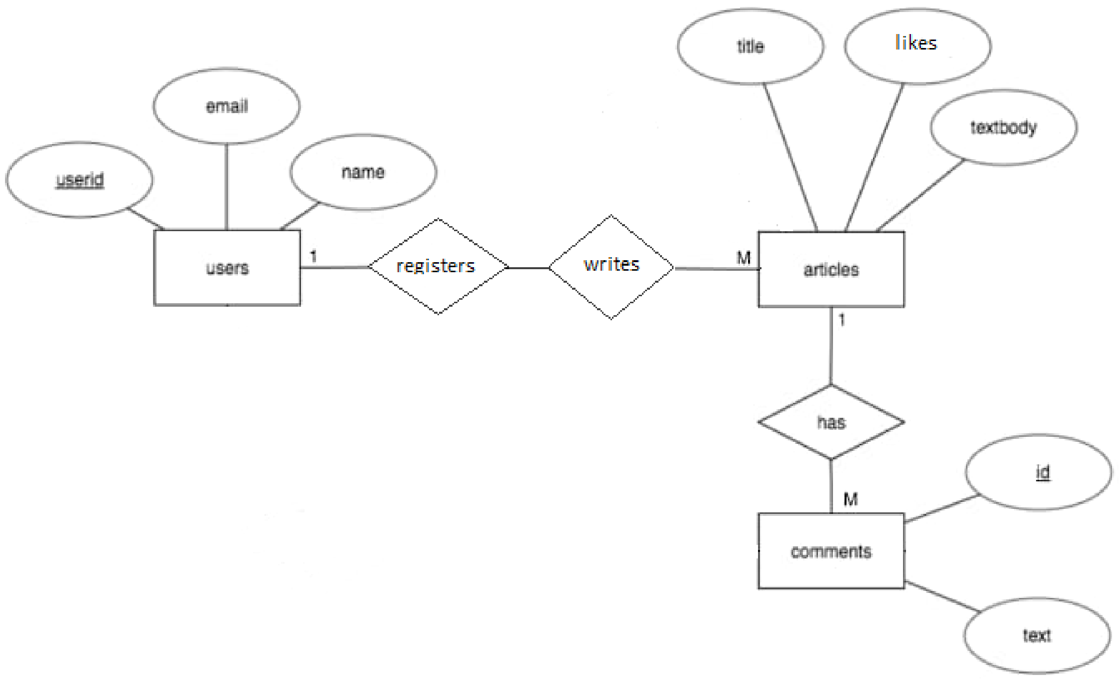


Fig 2: System design diagram

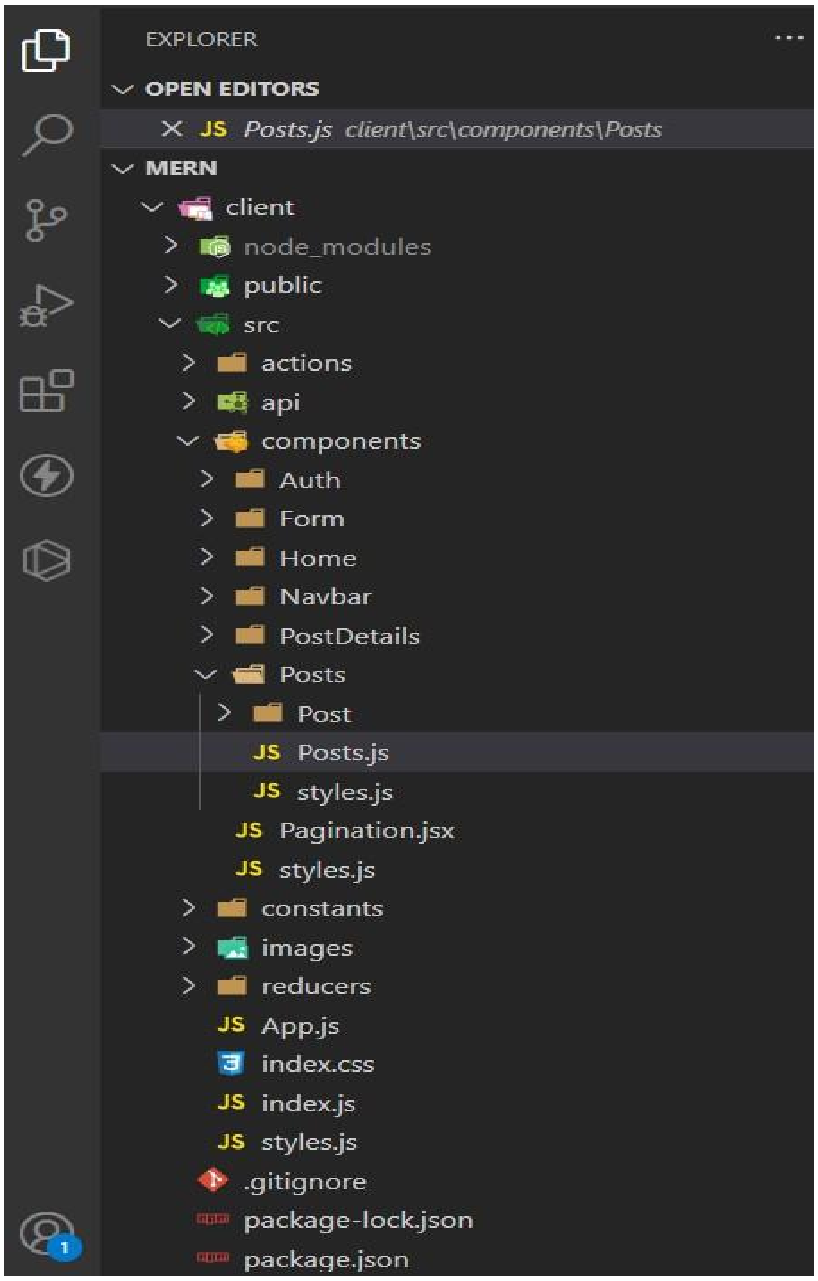
3.2 System design implementation

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.

3.2.1 Identification of features

- Creation of a post with a test body and image in jpg format.

- Updation of a post only by a user who is logged-in.
- Deletion of an existing entry only by the user who created it.
- Fetching a post based on title or hashtag.
- Creating a user (signup).
- Letting the user login.



```
EXPLORER
OPEN EDITORS
  × JS Posts.js client\src\components\Posts
MERN
  client
    node_modules
    public
    src
      actions
      api
      components
        Auth
        Form
        Home
        Navbar
        PostDetails
        Posts
          Post
          Posts.js
          styles.js
        Pagination.jsx
        styles.js
      constants
      images
      reducers
      App.js
      index.css
      index.js
      styles.js
    .gitignore
    package-lock.json
    package.json
```

Fig 3: MERN work environment

3.2.2 Libraries and frameworks used

JS is a lightweight, compiled or interpreted scripting language with many functions . Many non-browser contexts, like Node.js, Apache CouchDB, and Adobe Acrobat,use it as it is one of the best server-side scripting language.

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.

Nodejs : Open-source server environment Node.js is cross-platform and works with a variety of operating systems, including Windows, Linux, Unix, macOS, and more. JavaScript code can be executed outside of a web browser using Node.js, a back-end runtime environment that uses the V8 JavaScript Engine. JavaScript is a scripting language that developers may use to create server-side scripts and command-line tools using Node.js. In order to create dynamic web page content before the page is transmitted to the user's web browser, it is frequently necessary for the server to be able to run JavaScript code.

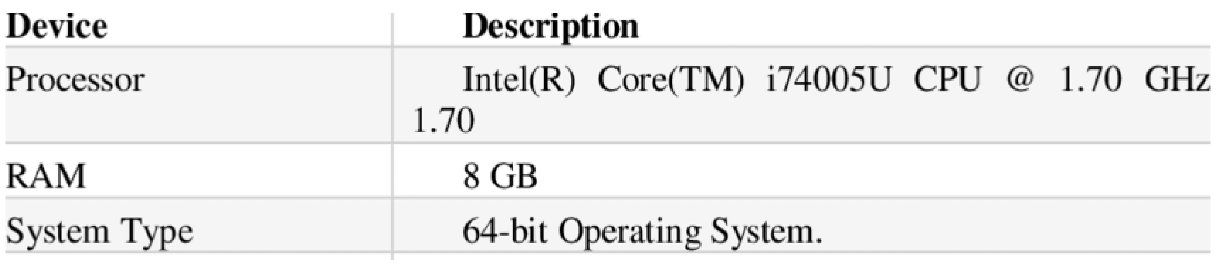
Mongoose : Mongoose is a mongodb library for writing concise and readable code. It handles data associations, offers validation, and translates between objects created in code and how MongoDB represents those same items. This indicates that Mongoose enables the definition of objects with strongly-typed 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.

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.

3.2.3 Technical Requirements

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

3.2.4 Hardware Configuration



Device	**Description**
Processor	Intel(R) Core(TM) i74005U CPU @ 1.70 GHz 1.70
RAM	8 GB
System Type	64-bit Operating System.

Table 1: Hardware requirements

3.2.5 Software Configuration

Operating System	**Windows**
Language	**Javascript/JSX**
Package manager	**Node package manager (NPM)**
Runtime environment	**Node.js**

Table 2: Software requirements

3.3 Website Design

Header containing the name of the website along with a navigation bar. The navigation bar has links to “about” , “articles” and “security” pages.

Body which has different contents for each page. Eg:- In the “articles” page body contains a search bar, article along with their links while in the “Security” page it has a login form.

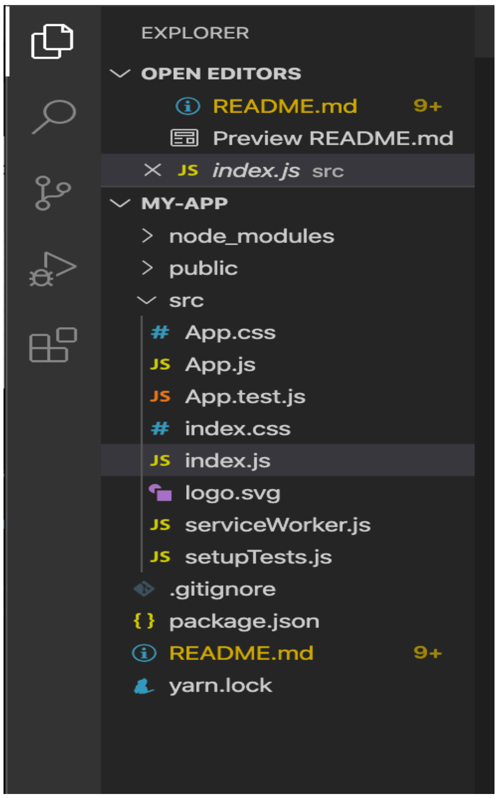
3.4 Front-end implementation

MOVING FORWARD WITH REACT

3.4.1 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.

3.4.2 React Workspace and folder hierarchy



```
EXPLORER
OPEN EDITORS
    README.md  9+
    Preview README.md
  × index.js src
MY-APP
  > node_modules
  > public
  ∨ src
      App.css
      App.js
      App.test.js
      index.css
      index.js
      logo.svg
      serviceWorker.js
      setupTests.js
    .gitignore
    package.json
    README.md  9+
    yarn.lock
```

Fig 4: Bootstrapped react app made using npm create-react-app command

3.4.3 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.

3.4.4 Importing and exporting components

In react we can export a particular component and then import it in a parent component to reuse it multiple times.

Importing :

Importing a component - import Gallery from './Gallery.js';

Importing dependencies - import axios from ‘axios’;

Importing hooks - import {useState} from ‘react’;

Exporting :

We can export a component as follows :

```
export default function App() {
  return (
    <Gallery />
  );
}
```

3.4.5 How react works (JSX and Babel) :

JSX stands for JavaScript XML. JSX allows you to write html inside javascript. It is this feature that makes react so powerful and clean.

An expression in JSX : `const myElement = <h1>Current version of React is {9+9}.</h1>;`

A block of html in react can be written as :

```
const myElement = (
  <div>
    <p>First paragraph.</p>
    <p>Second paragraph</p>
  </div>
```

What is Babel ?

Founded by Sebastian Mckenzie, BabelJS is a JavaScript transpiler that converts new features into out-of-date norms. With this, it is simple to use the functionality on both old and modern browsers. It is used to compile JSX which is used by react.

The language that the browser comprehends is JavaScript. To run our applications, we use a variety of browsers, including Chrome, Firefox, Internet Explorer, Microsoft Edge, Opera, and UC Browser. The JavaScript language specification is known as ECMA Script; the most recent stable version, ECMA Script 2015 ES6, is compatible with both new and old browsers.

We've had ES6, ES7, and ES8 since ES5. There are numerous new features in ES6 that not all browsers fully support. The same holds true for ES7, ES8, and ESNext (the upcoming ECMA Script version). When all browsers will be able to work with every ES version that was published is currently unknown.

We require a programme that will compile our final code in ES5 if we want to use new ECMAScript capabilities and run it on every available browser. Babel is used to resolve this issue.

3.4.6 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" };
   }
}
```

3.4.7 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'.

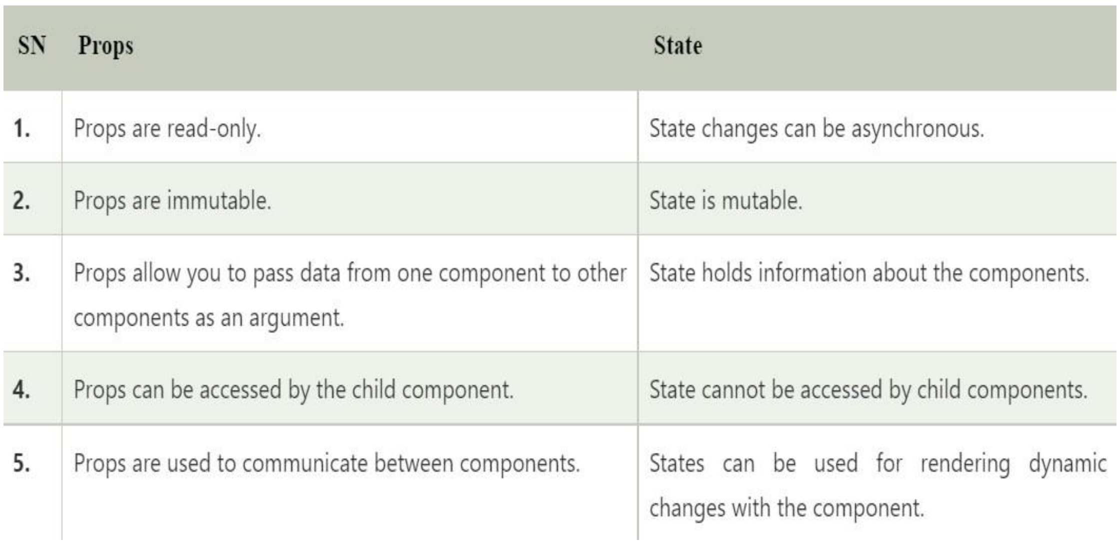
Eg:-

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.

3.4.8 React Hooks

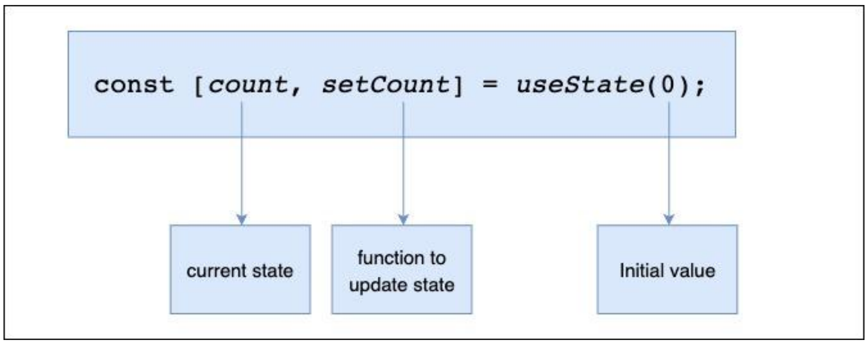
React version 16.8 introduced hooks. It was done to replace the class components. Hooks allow us to access the state of a component and other react features. States can be changed via hooks. It also helps us to 'hook' into the lifecycle methods. The programmer needs to import the hook before using it. Hooks can be imported by the following line of code :

- import {useState} from 'react';

Below is the list of the most frequently used react hooks :

1. **useState hook :** used to set and modify the state of a component.

   Used to change / update the state of a component. Basically if anything changes in a react application, its state is said to be changed. If we type anything in a search bar, its state changes and everytime a state changes, the page reloads.



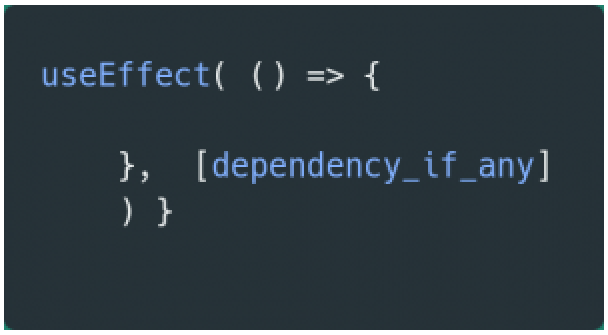
```
const [count, setCount] = useState(0);
```

Fig 5 : Working of useState hook

2. **useEffect hook :** used to perform a specific task once an event is triggered.

   useEffect hook takes 2 arguments: an anonymous function and an array. The code inside the anonymous function will be executed once the argument in the array is triggered. Here we are increasing the value of the count variable once a certain state changes.

   Syntax of useEffect hook :



```
useEffect( () => {

    },  [dependency_if_any]
    ) }
```

3. **useContext hook :** used to avoid prop drilling. A context has states which can be accessed by any component no matter how many parent components are present above it.

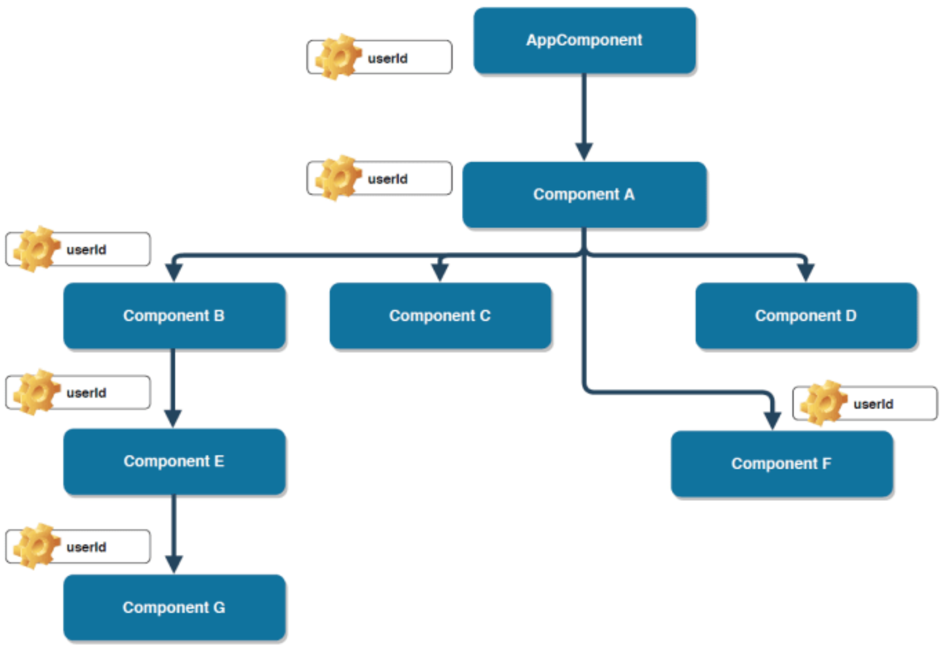


Fig 6 : Working of useContext hook

4. **useRef hook :** Introduces the concept of uncontrolled components. Uncontrolled components are not controlled by react state but by the DOM itself. Refs are used to interact directly with real DOM(document object model) not the virtual one. Refs don't cause re-renders. Suppose we want to use the 'focus' property which is present in javascript but not in JSX. By using Refs, we can get hold of the original DOM node and use the focus property on it. When a button is clicked and you want an input to become the focus as a result, this is a very typical use case for useRef. In order to accomplish this, we would first need to access the input DOM element and then call its focus() function. To accomplish this with JavaScript, all that is required is to choose the input using the querySelector or by id/class, after which the focus() function is called. However, React does not come with a built-in method for accomplishing this.

5. **useMemo hook :** used to stop the unnecessary re-renders. When the state of a parent component changes, the child component also has to re-render despite the fact that it's state hasn't been changed. This causes unnecessary memory loss. This behaviour can be prevented by useMemo hook. Consider a static Welcome Card that will be shown inside an application. Other states, such a counter, are also included in the application. The Welcome Card is a child of the primary parent App, therefore once the counter is raised, the static card will be updated as a result of any changes to the app's internal state.

6. **useCallback hook :** Similar to useMemo. It returns the memoized function while useMemo returns the memoized value. UseMemo doesn't work when a prop is passed from parent to child. In such cases useCallback hook is preferred.

   Pass an array of dependencies along with an inline callback. A memoized version of the callback that only changes if one of the dependencies has changed is what useCallback returns. This is helpful for sending callbacks to optimised child components that don't need to render everything because they rely on reference equality.

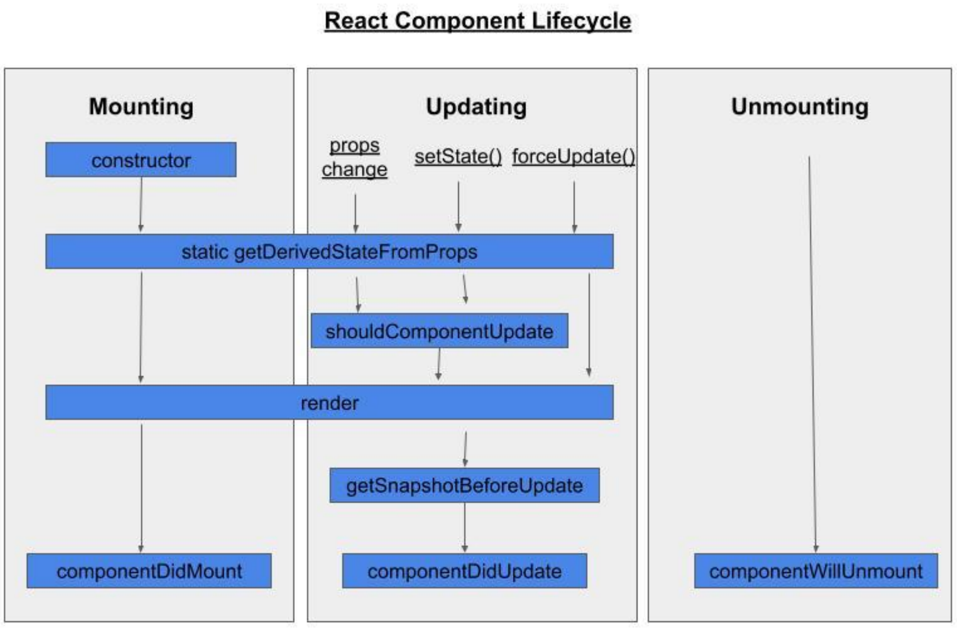


Fig 7 : React hooks lifecycle

Mounting, updating, and unmounting are the three phases that make up a React component's lifetime. Each phase has unique methods in charge of a certain stage in a component's lifetime. Technically speaking, these methods are not intended for functional components and are only applicable to class-based components.

However, because the Hooks concept was introduced in React, you can now use abstracted variations of these lifecycle methods when working with functional component state. Simply put, React Hooks are functions that enable "hooking into" React states and lifecycle elements from within function components.

Phases of React component lifecycle :

A new component is produced and added to the DOM during the mounting step, which also marks the start of a component's life. This is frequently referred to as the "initial render" and can only occur once.

The component updates or re-renders during the updating process. When the state or the props are updated, this reaction is triggered. Multiple occurrences of this phase are possible, which is sort of React's purpose.

The unmounting step, in which the component is taken out of the DOM, is the final stage of a component's lifetime.

For each phase of the life cycle, a class-based component can call a different method (more on this below). These lifecycle methods, which can only be produced by or included in classes, are obviously not used by functional components. On the other hand, when employing React hooks, functional components can benefit from states.

3.4.9 Styling using Material-ui

It is one of the most popular React-ui component frameworks maintained by the react community. Google's Material Design is implemented through a free and open-source React component bundle called Material UI. It comes with a wide range of prebuilt components that are employed right away in production.

Installation : npm install @material-ui/core

We use material-ui by making an object for styling a particular component in a separate file usually named 'styles.css' and the using the styles inside the component as follows :

Initialising Class using useStyle hook : const classes = useStyles();

Providing style to a component : <Avatar className={classes.avatar}>

Here 'avatar' is an object present in the 'styles.css' page.

3.5 State Management using React-redux :

Redux is one of the most important tools present in react. It is an open-source javascript library used for storing all the states at a centralised location. It is mainly used for state management. When there are a large number of states, redux stores all of them at a centralised location called Redux Store. From there, the 'actions' are dispatched to update the data. Redux provides a simplified and sensible way to manage the states. It works on the principle of 'unidirectional data flow'. It helps to scale the application and manage it more efficiently.

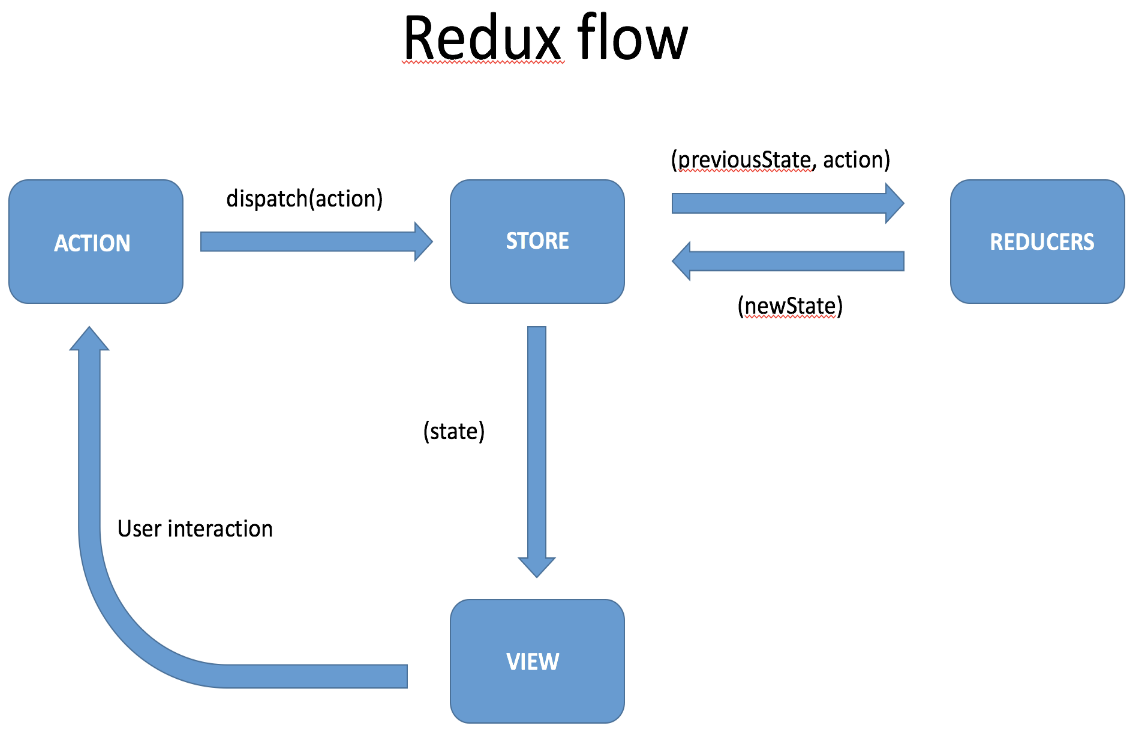


Fig 8 : The flow control of redux

The three main components of Redux are :

Redux Store : Stores the current states of all the components. Whenever the data is needed, we need to access the Store. The reducers then update the data and an action is dispatched. The updated data is then stored in the Redux Store.

Creating a Redux Store :

```
const store = createStore(reducers)
ReactDOM.render(
   <Provider store={store}>
       <App />
   </Provider>,
 document.getElementById('root')
);
```

Action : Actions are straightforward objects with the typical two properties of type and payload. The payload property is an optional property that holds some data that is necessary to complete any given task, whereas the type property is often a text that specifies the action. Sending information from the application to the Redux store is the primary purpose of an action.

Reducers : Reducers are pure functions that adjust the application's state in response to user input. Reducers accept an action and a previous state as input and output a modified state. Due to the immutability of the state, a reducer always produces a new state that represents an updated version of the prior state. Basically it is used to perform some action and update data/state present in the Redux Store.

```
case AUTH:
     localStorage.setItem('profile', JSON.stringify({ ...action?.data }));
     return { ...state, authData:action?.data };

case LOGOUT:
     localStorage.clear();
     return { ...state, authData: null, loading: false, errors: null };
```

Here the reducers are updating the state when a user tries to login and logout.

Data flow in a Redux application :

- The flow of data is triggered when a user interacts with a component. The action creators dispatch an ‘action’ due to this interaction.
- Once an action is dispatched, it is received by the application's root reducer and distributed to all other reducers. Therefore, based on the dispatched action, it is the reducer's responsibility to decide whether it needs to update the state.
- This is verified by filtering out the necessary actions using a straightforward switch statement. Each (smaller) reducer in the application accepts the dispatched action and returns a newly updated state if the type of the dispatched action matches.
- It is important to remember that with redux, the state never actually changes. Instead, the reducer always creates a new state that is an exact duplicate of the old state but has undergone some changes.
- The component is then notified by the store of the altered state, which causes it to retrieve it and render the component again.
- The fact that data flow in a React-Redux application is unidirectional, or only going in one direction, is another crucial finding in this context.

3.6 Routing

A frequently used tool for making custom routes in React is React Router. It allows switching between multiple pages made by various React components, permits changing the browser's URL and maintains UI synchronisation with the URL.

Installation : npm install react-router-dom

Importing :

```
 import {
        BrowserRouter as Router,
        Routes,
        Route,
        Link
} from 'react-router-dom';
```

Usage : This application has five main routes. Home, Auth, Posts, Search, :id. The exact paths are as follows :

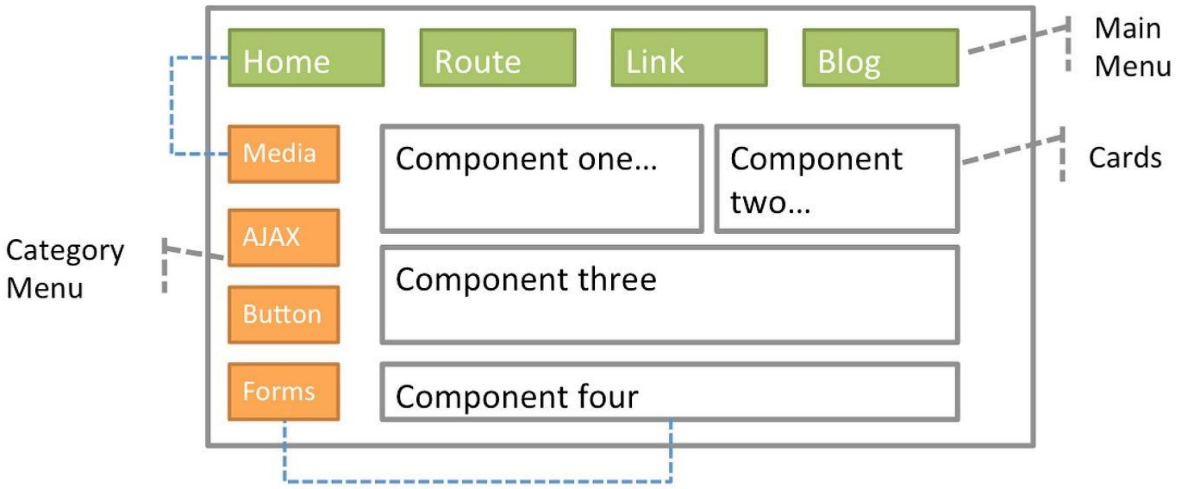


Fig 9 : Working of react router

3.7 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 :

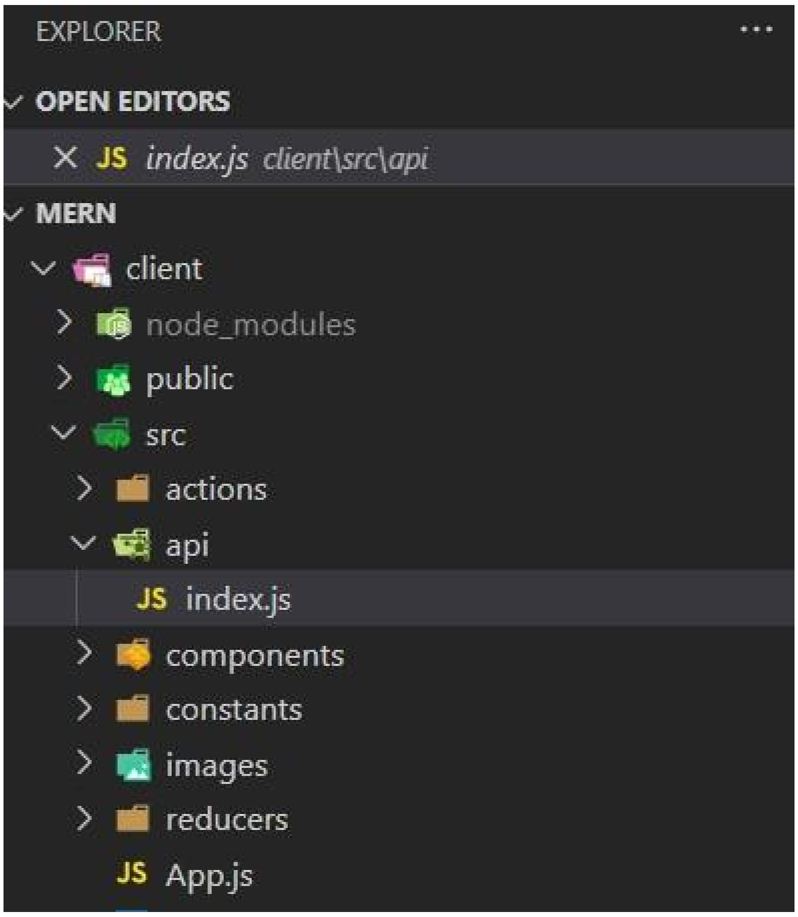


Fig 10 : Location of API file

Axios : We are using the ‘axios’ library to make our custom API. Through Axios, which also supports the Promise API that is part of JS ES6, we connect with the backend. It is a package that enables us to request information from APIs, receive that information, and use it to carry out activities in our React apps.

Importing : import axios from ‘axios’;

Providing the URL : Our application is currently hosted locally on localhost. So the link is :

```
const API = axios.create({ baseurl: 'http://localhost:5000'})
```

Using Axios :

APIs for Signin / Signup : Sending post request to the backend (the specified Url)

```
export const signIn = (formData) => API.post('/user/signin', formData);

export const signUp = (formData) => API.post('/user/signup', formData);
```

APIs for updating a particular post : Like, Update and Delete

```
export const likePost = (id) => API.patch(`/posts/${id}/likePost`);

export const updatePost = (id, updatedPost) => API.patch(`/posts/${id}`,
updatedPost);

export const deletePost = (id) => API.delete(`/posts/${id}`);
```

We send a ‘patch’ request whenever we need to update an existing post.

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.

3.8 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 :

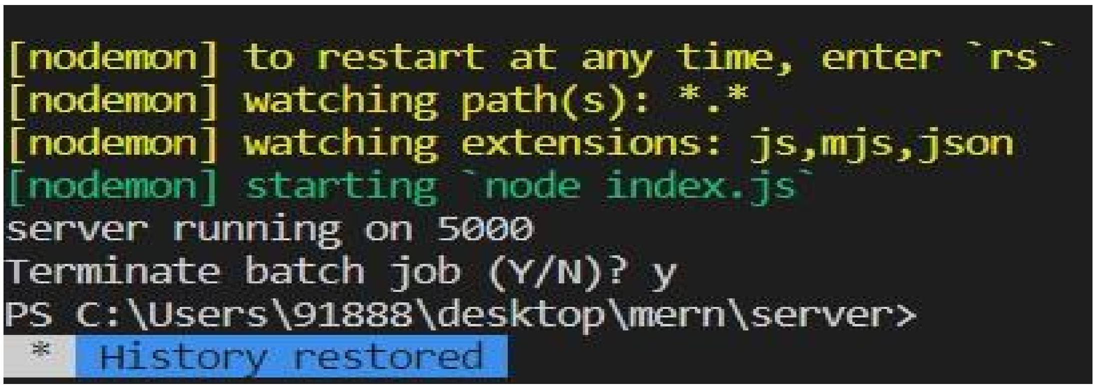
```
const app = express();

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 :

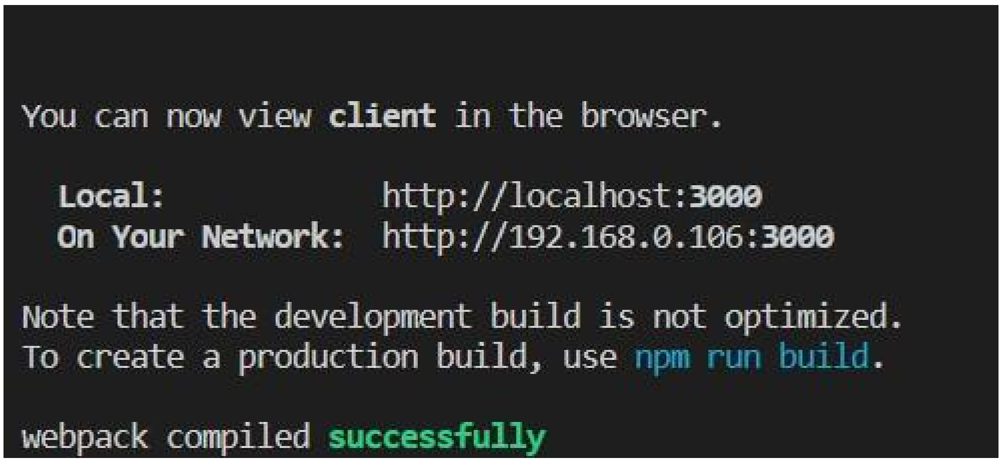
```
const PORT = process.env.PORT || 5000;

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 5000. This means that our client runs on localhost: 3000 while server runs on localhost: 5000. After these set-ups, we can start our app by running ‘npm start’ command :



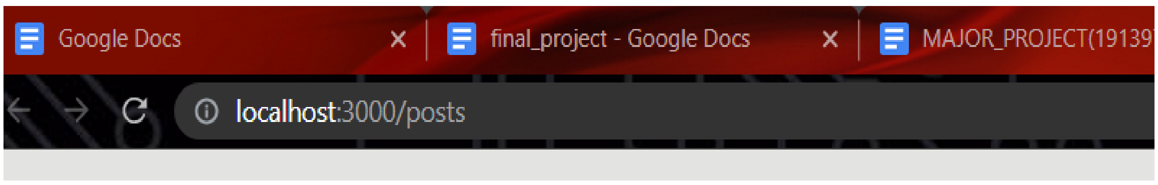
```
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
server running on 5000
Terminate batch job (Y/N)? y
PS C:\Users\91888\desktop\mern\server>
 *  History restored
```

Fig 11 : Server running on port 5000



```
You can now view client in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.106:3000

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

webpack compiled successfully
```

Fig 12 : Client running on port 3000



```
Google Docs    final_project - Google Docs    MAJOR_PROJECT(19139
localhost:3000/posts
```

Fig 13 : The url of any page starts with localhost:3000

3.9 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 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.

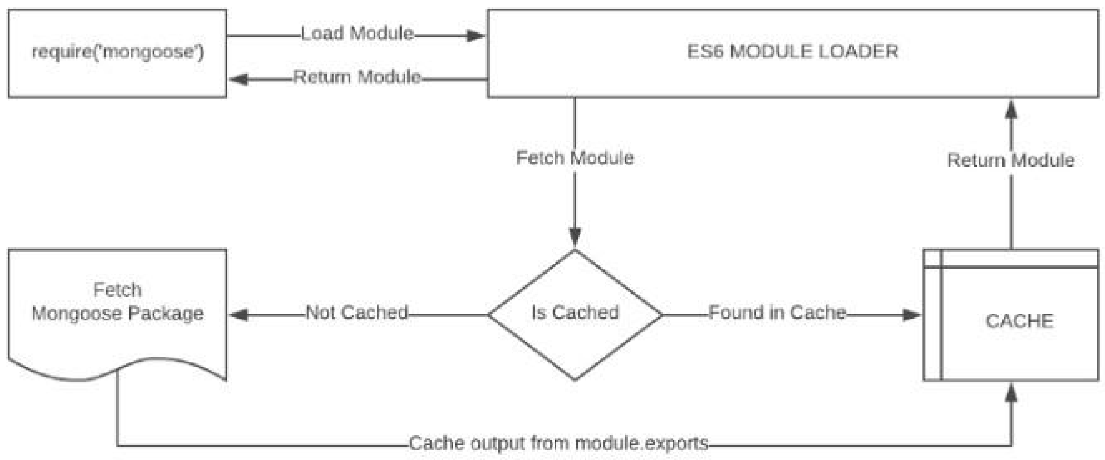


Fig 14 : Module import/require workflow

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.

THE USER SCHEMA :

```
import mongoose from "mongoose";

const userSchema = mongoose.Schema({
  name: { type: String, required:  true },

  email: { type: String, required: true },

  password: { type: String, required: true },

  id: {

        type: String

        },

});

export default mongoose.model("User", userSchema);
```

Here we are exporting the 'User' model based on the 'userSchema'. 'User' is automatically converted to plural form by mongoose.

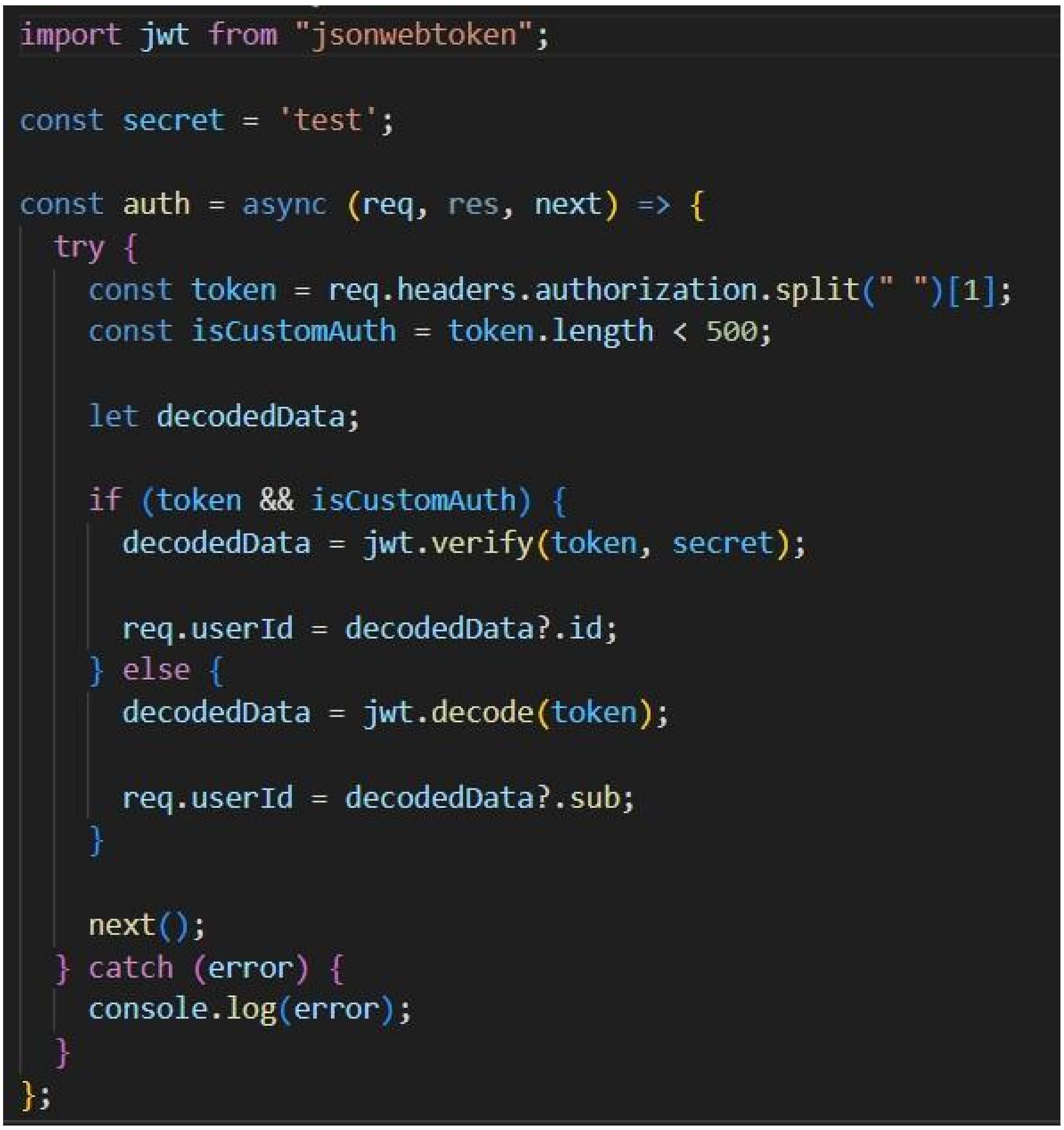
THE POST SCHEMA :

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.

```
const postSchema = mongoose.Schema({
   posttitle: String,
   msg: String,
   username: String,
   creator: String,
   hash : [String],
   selectedFile: String,
   numberoflikes : {
       type: [String],
       default: [],
   },
   time : {
       type: Date,
       default: new Date()
   },
});

const PostMessage = mongoose.model('PostMessage', postSchema);
export default PostMessage;
```

MIDDLEWARE FOR AUTHENTICATION :



```
import jwt from "jsonwebtoken";

const secret = 'test';

const auth = async (req, res, next) => {
  try {
    const token = req.headers.authorization.split(" ")[1];
    const isCustomAuth = token.length < 500;

    let decodedData;

    if (token && isCustomAuth) {
      decodedData = jwt.verify(token, secret);

      req.userId = decodedData?.id;
    } else {
      decodedData = jwt.decode(token);

      req.userId = decodedData?.sub;
    }

    next();
  } catch (error) {
    console.log(error);
  }
};
```

Fig 15 : Middleware code for authentication

3.10 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 :

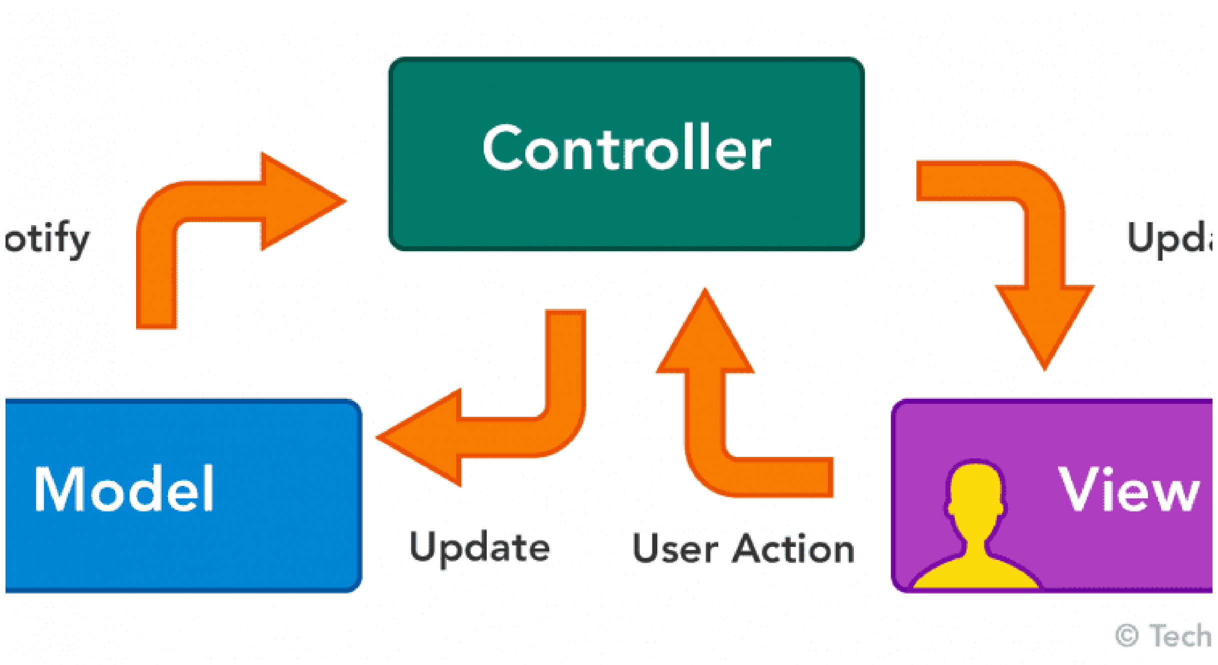


Fig 16 : Controllers

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.

2. Updating a post

```
export const updatePost = async (req, res) => {
    const {id: _id} = req.params;
    const post = req.body;
    if(!mongoose.Types.ObjectId.isValid(_id)) return res.status(404).send('No post found');
     const updatedPost = await PostMessage.findByIdAndUpdate(_id, {...post, _id}, {new: true});
    res.json(updatedPost);
}
```

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 it's id inside the database.

3. Creating a post

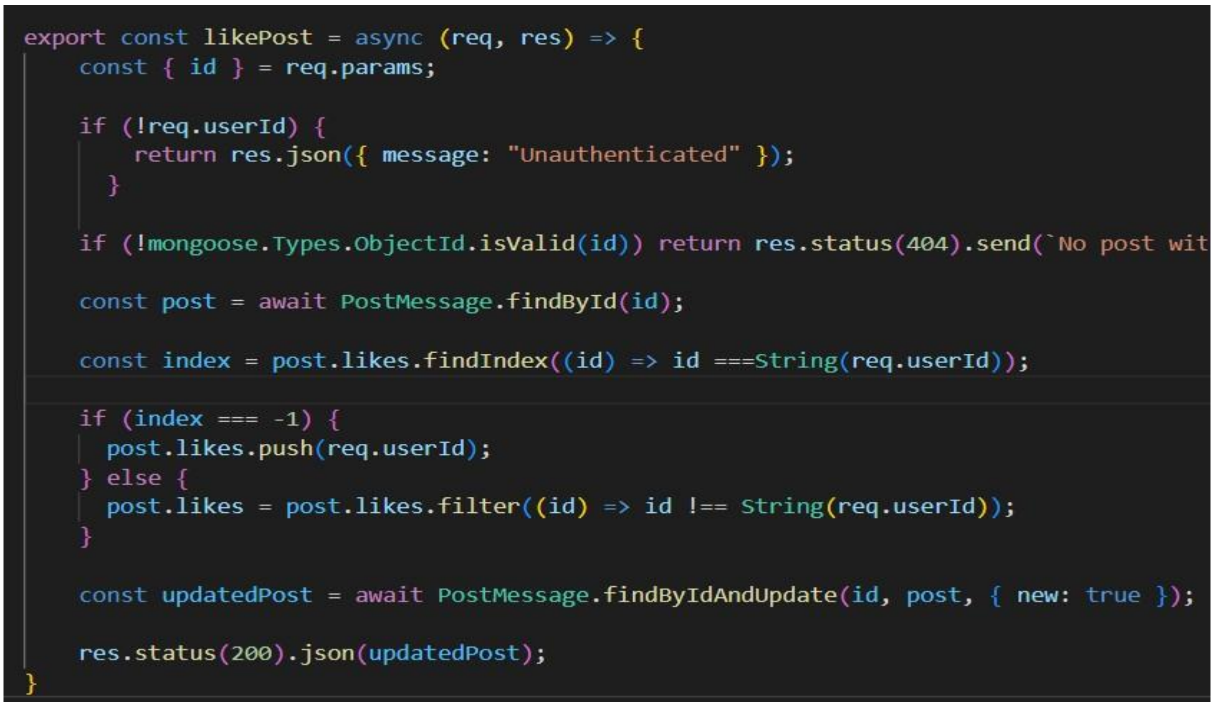
```
export const createPost = async (req, res) => {
    const post = req.body;
     const newPostMessage = new PostMessage({ ...post, creator: req.userId, createdAt: new Date().toISOString() })
    try {
        await newPostMessage.save();
        res.status(201).json(newPostMessage );
    } catch (error) {
        res.status(409).json({ message: 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.

5. Liking a post



```
export const likePost = async (req, res) => {
    const { id } = req.params;

    if (!req.userId) {
        return res.json({ message: "Unauthenticated" });
      }

    if (!mongoose.Types.ObjectId.isValid(id)) return res.status(404).send(`No post wit

    const post = await PostMessage.findById(id);

    const index = post.likes.findIndex((id) => id ===String(req.userId));

    if (index === -1) {
      post.likes.push(req.userId);
    } else {
      post.likes = post.likes.filter((id) => id !== String(req.userId));
    }

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

    res.status(200).json(updatedPost);
}
```

Fig 17 : Liking a post

The Security Controller

1.Signin Controller

```
export const signin = async (req, res) => {
  const { email, password } = req.body;
  try {
    const oldUser = await UserModal.findOne({ email });
    if (!oldUser) return res.status(404).json({ message: "User doesn't exist" });
    const isPasswordCorrect = await bcrypt.compare(password, oldUser.password);
    if (!isPasswordCorrect) return res.status(400).json({ message: "Invalid credentials" });
    const token = jwt.sign({ email: oldUser.email, id: oldUser._id }, secret, { expiresIn: "1h" 
});
    res.status(200).json({ result: oldUser, token });
  } catch (err) {
    res.status(500).json(
         { message: "Something went wrong" }
         );
  }
};
```

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.

3.11 Overall flow of data in the web application :

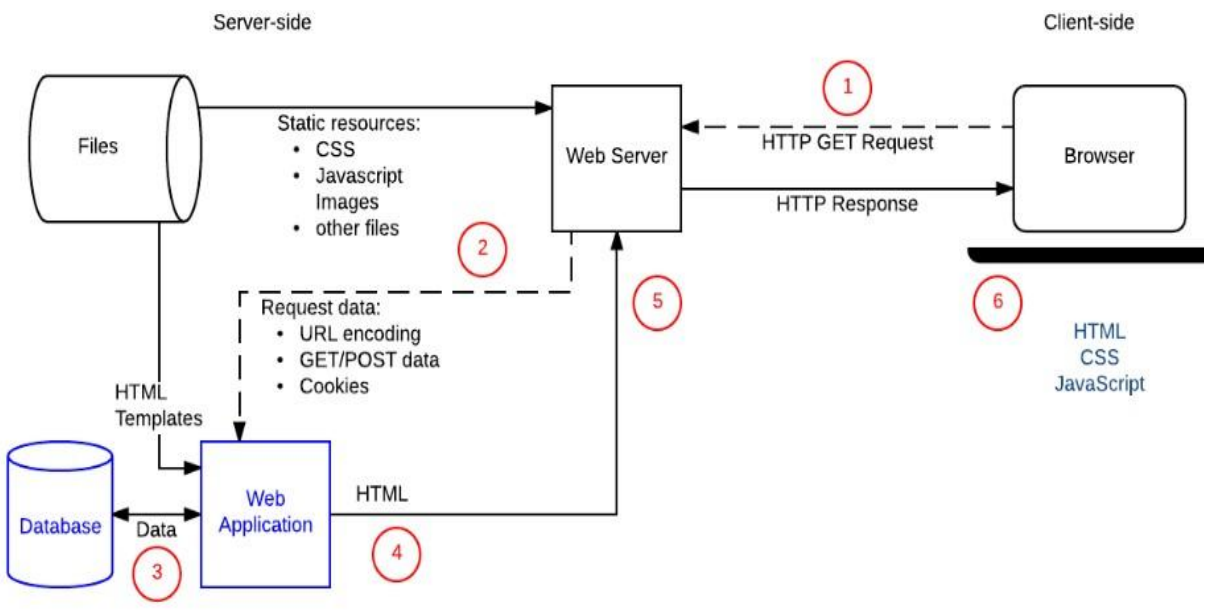


Fig 18 : data flow in a typical web application

3.12 Three-tier architecture

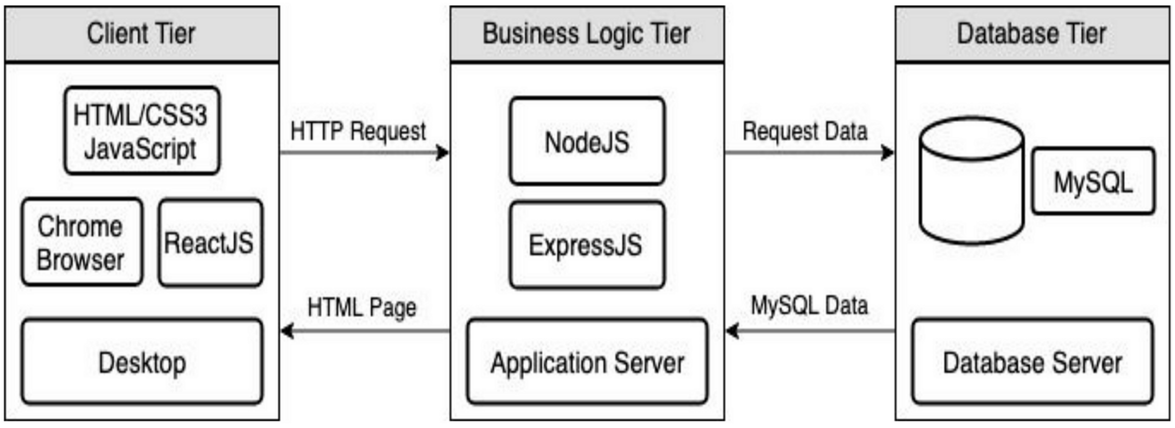


Fig 19 : The web dev architecture(three-tier)

Data tier : The application manages and stores the information it processes. This could be a NoSQL database server like Cassandra, CouchDB, or MongoDB, or a relational database management system like PostgreSQL, MySQL, MariaDB, Microsoft SQL Server.

Benefits of three-tier architecture :

1. Faster development as multiple developers can work on different tiers.
2. Improved Scalability for the same reasons mentioned above.
3. Improved Reliability
4. Improved Security.

4. RESULT

4.1 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.

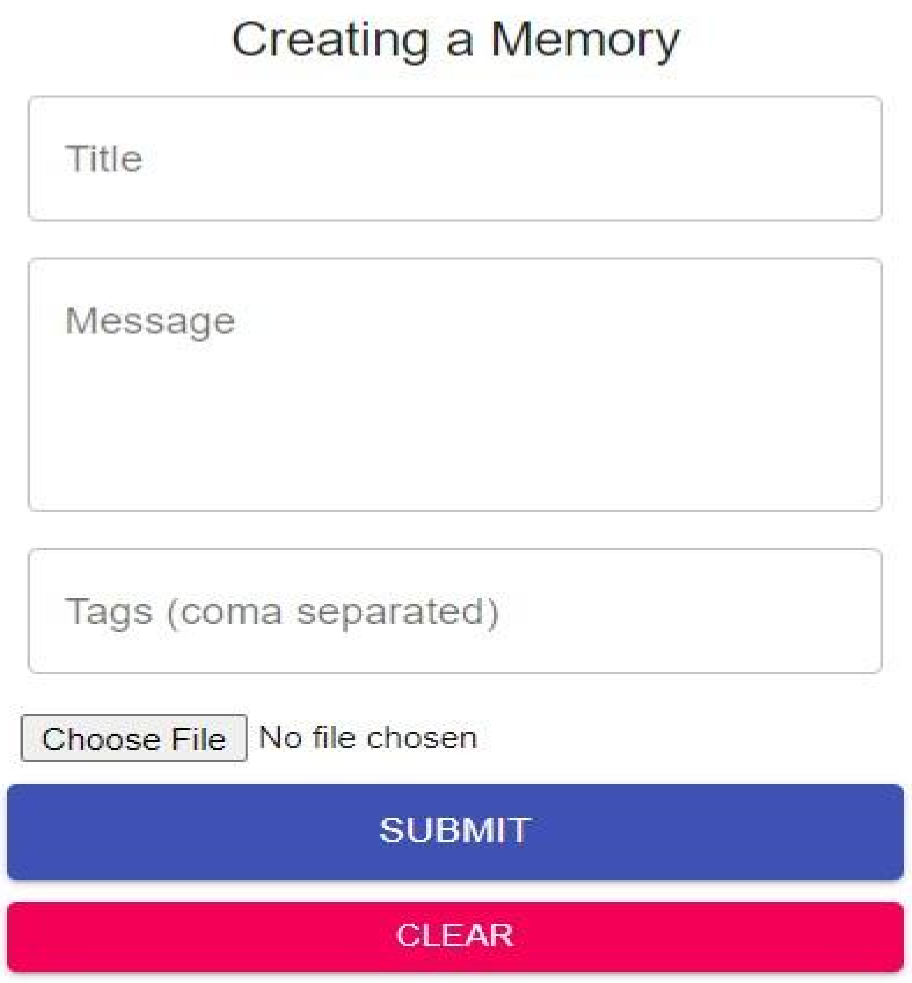
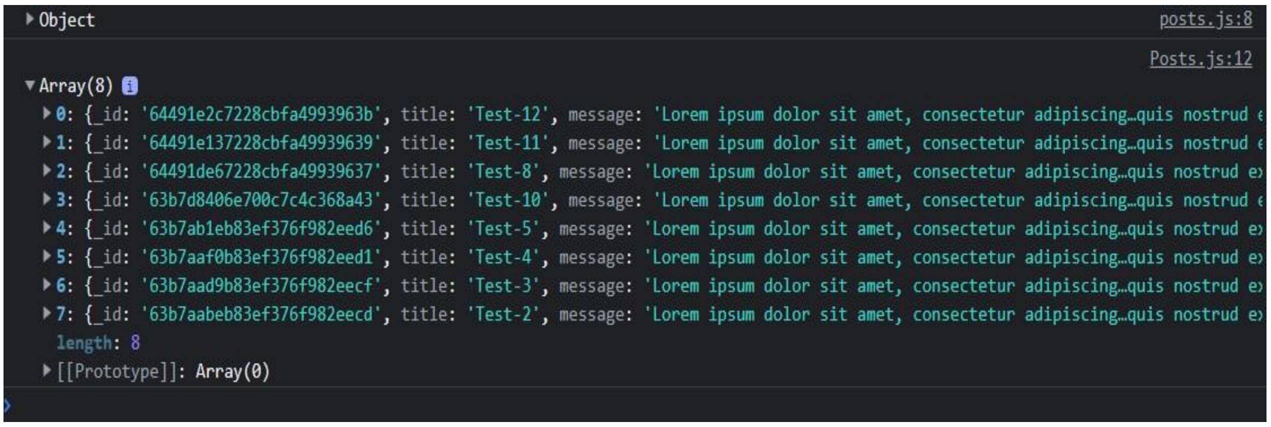


Fig 20 : Post creation column design

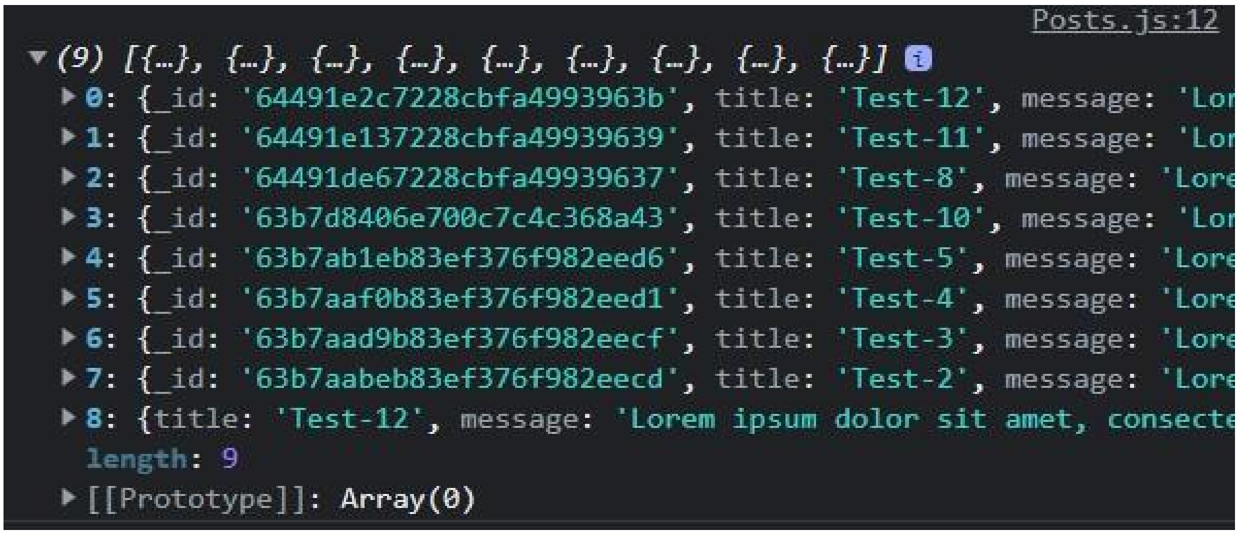
The submit button sends a request to the database to save the new post and the user credentials through the API. The clear button will clear all the fields at once.



```
▸Object                                                                                                    posts.js:8
                                                                                                           Posts.js:12
▾Array(8) 
  ▸0: {_id: '64491e2c7228cbfa4993963b', title: 'Test-12', message: 'Lorem ipsum dolor sit amet, consectetur adipiscing…quis nostrud e
  ▸1: {_id: '64491e137228cbfa49939639', title: 'Test-11', message: 'Lorem ipsum dolor sit amet, consectetur adipiscing…quis nostrud e
  ▸2: {_id: '64491de67228cbfa49939637', title: 'Test-8', message: 'Lorem ipsum dolor sit amet, consectetur adipiscing…quis nostrud ex
  ▸3: {_id: '63b7d8406e700c7c4c368a43', title: 'Test-10', message: 'Lorem ipsum dolor sit amet, consectetur adipiscing…quis nostrud e
  ▸4: {_id: '63b7ab1eb83ef376f982eed6', title: 'Test-5', message: 'Lorem ipsum dolor sit amet, consectetur adipiscing…quis nostrud ex
  ▸5: {_id: '63b7aaf0b83ef376f982eed1', title: 'Test-4', message: 'Lorem ipsum dolor sit amet, consectetur adipiscing…quis nostrud ex
  ▸6: {_id: '63b7aad9b83ef376f982eecf', title: 'Test-3', message: 'Lorem ipsum dolor sit amet, consectetur adipiscing…quis nostrud ex
  ▸7: {_id: '63b7aabeb83ef376f982eecd', title: 'Test-2', message: 'Lorem ipsum dolor sit amet, consectetur adipiscing…quis nostrud ex
   length: 8
  ▸[[Prototype]]: Array(0)
>
```

Fig 21 : First 8 posts appears on the first page

Above is the console view of the first eight 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', title, message, tags, etc.



```
                                                                              Posts.js:12
▾(9) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}] 
  ▸0: {_id: '64491e2c7228cbfa4993963b', title: 'Test-12', message: 'Lor
  ▸1: {_id: '64491e137228cbfa49939639', title: 'Test-11', message: 'Lor
  ▸2: {_id: '64491de67228cbfa49939637', title: 'Test-8', message: 'Lore
  ▸3: {_id: '63b7d8406e700c7c4c368a43', title: 'Test-10', message: 'Lor
  ▸4: {_id: '63b7ab1eb83ef376f982eed6', title: 'Test-5', message: 'Lore
  ▸5: {_id: '63b7aaf0b83ef376f982eed1', title: 'Test-4', message: 'Lore
  ▸6: {_id: '63b7aad9b83ef376f982eecf', title: 'Test-3', message: 'Lore
  ▸7: {_id: '63b7aabeb83ef376f982eecd', title: 'Test-2', message: 'Lore
  ▸8: {title: 'Test-12', message: 'Lorem ipsum dolor sit amet, consecte
   length: 9
  ▸[[Prototype]]: Array(0)
```

Fig 22 : New post appears as 9th element in the array

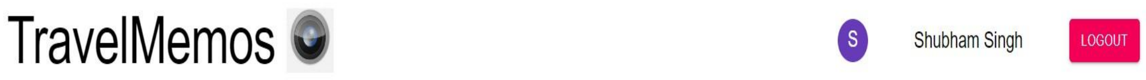


Fig 23 : The Navigation menu

On clicking the website logo, the user will be redirected to the home page. Other than that, we have a profile section and a logout button if the user is logged in otherwise a login button will appear.

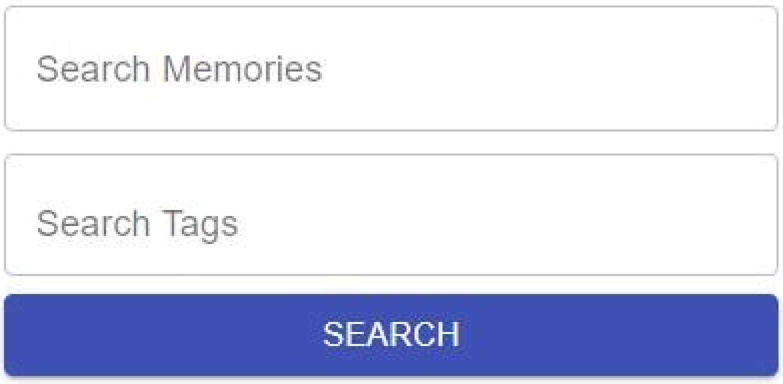
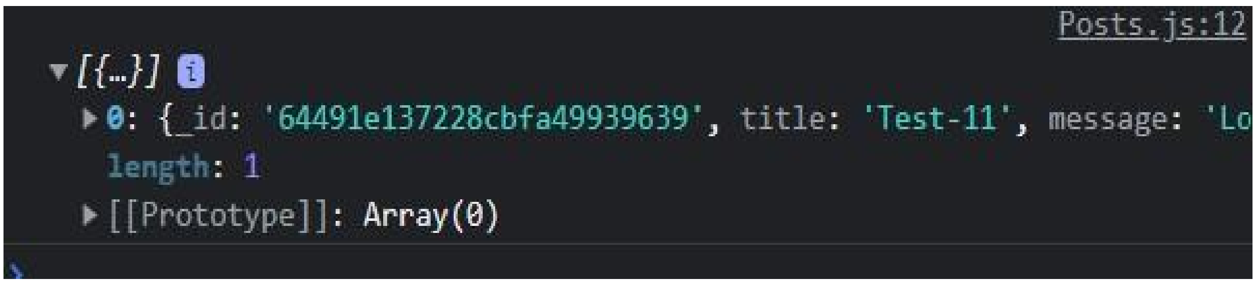


Fig 24 : Search bar

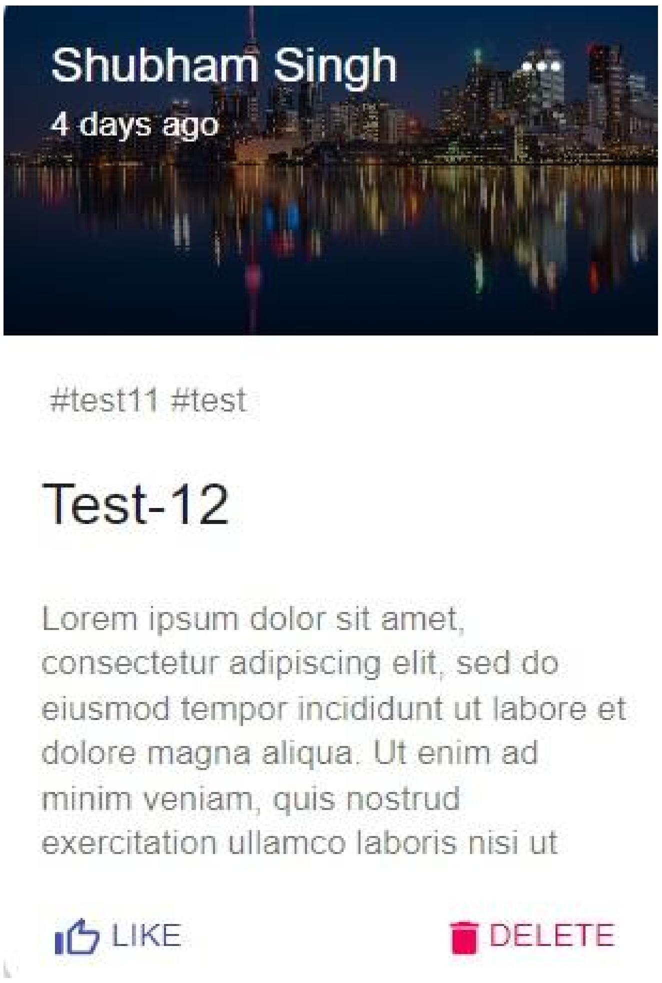
The users can search using either the title of the post or the tags of the post. Searching using the tags will have multiple results while the title will yield a specific post as the result.



```
                                                                Posts.js:12
▼[{…}] i
  ▶0: {_id: '64491e137228cbfa49939639', title: 'Test-11', message: 'Lo
   length: 1
  ▶[[Prototype]]: Array(0)
```

Fig 25 : Result of a search by title

4.2 A SPECIFIC POST :


Shubham Singh

4 days ago

#test11 #test

Test-12

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut

Fig 26 : A specific post

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.

4.3 The Security form design and console view :

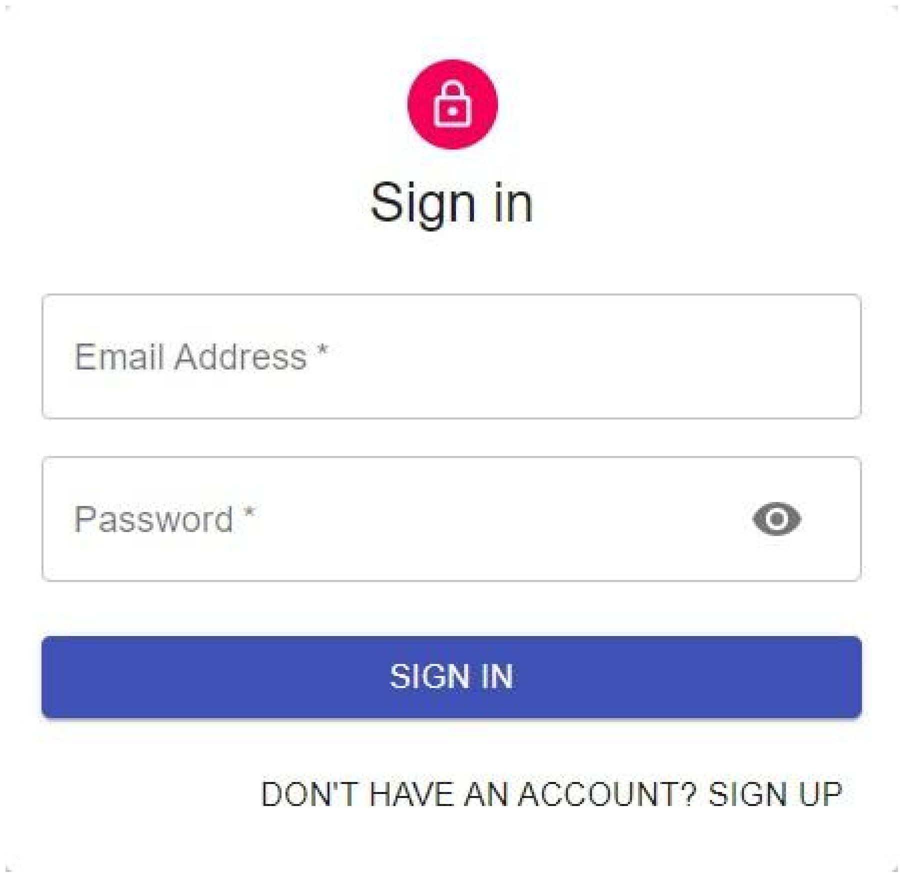
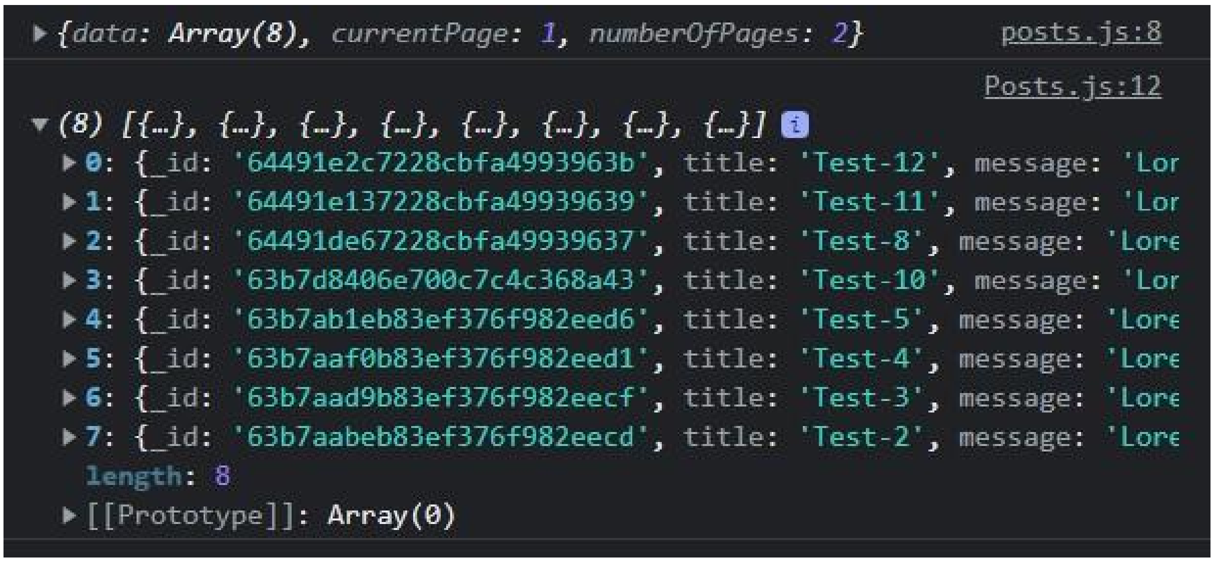


Fig 27 : The sign in column



```
▶{data: Array(8), currentPage: 1, numberOfPages: 2}          posts.js:8
                                                             Posts.js:12
▼(8) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}] ℹ
  ▶0: {_id: '64491e2c7228cbfa4993963b', title: 'Test-12', message: 'Lor
  ▶1: {_id: '64491e137228cbfa49939639', title: 'Test-11', message: 'Lor
  ▶2: {_id: '64491de67228cbfa49939637', title: 'Test-8', message: 'Lore
  ▶3: {_id: '63b7d8406e700c7c4c368a43', title: 'Test-10', message: 'Lor
  ▶4: {_id: '63b7ab1eb83ef376f982eed6', title: 'Test-5', message: 'Lore
  ▶5: {_id: '63b7aaf0b83ef376f982eed1', title: 'Test-4', message: 'Lore
  ▶6: {_id: '63b7aad9b83ef376f982eecf', title: 'Test-3', message: 'Lore
  ▶7: {_id: '63b7aabeb83ef376f982eecd', title: 'Test-2', message: 'Lore
   length: 8
  ▶[[Prototype]]: Array(0)
```

Fig 28 : The posts appear after signing in

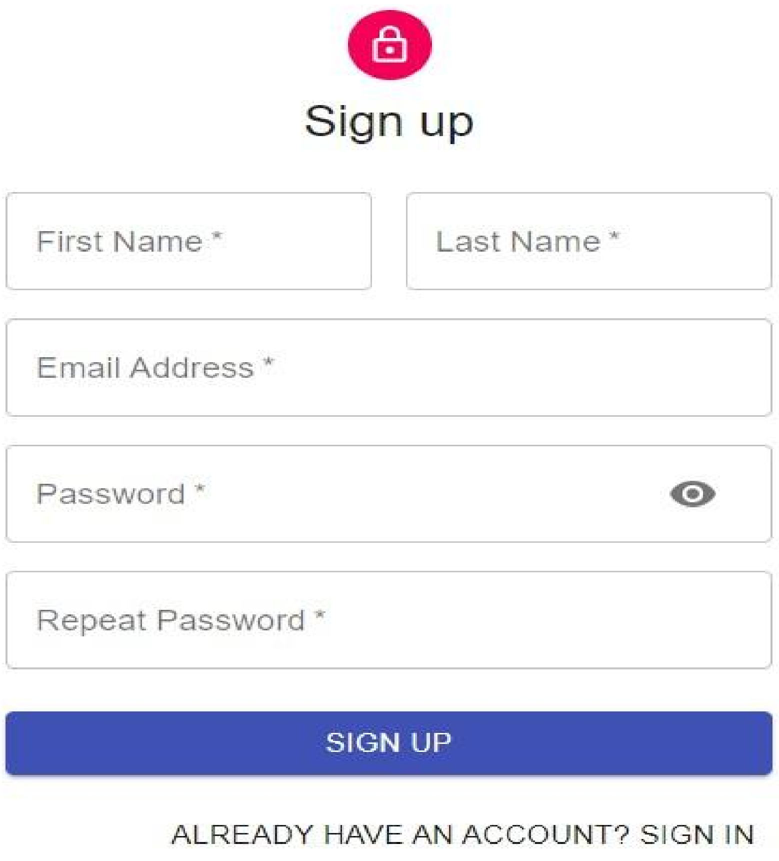
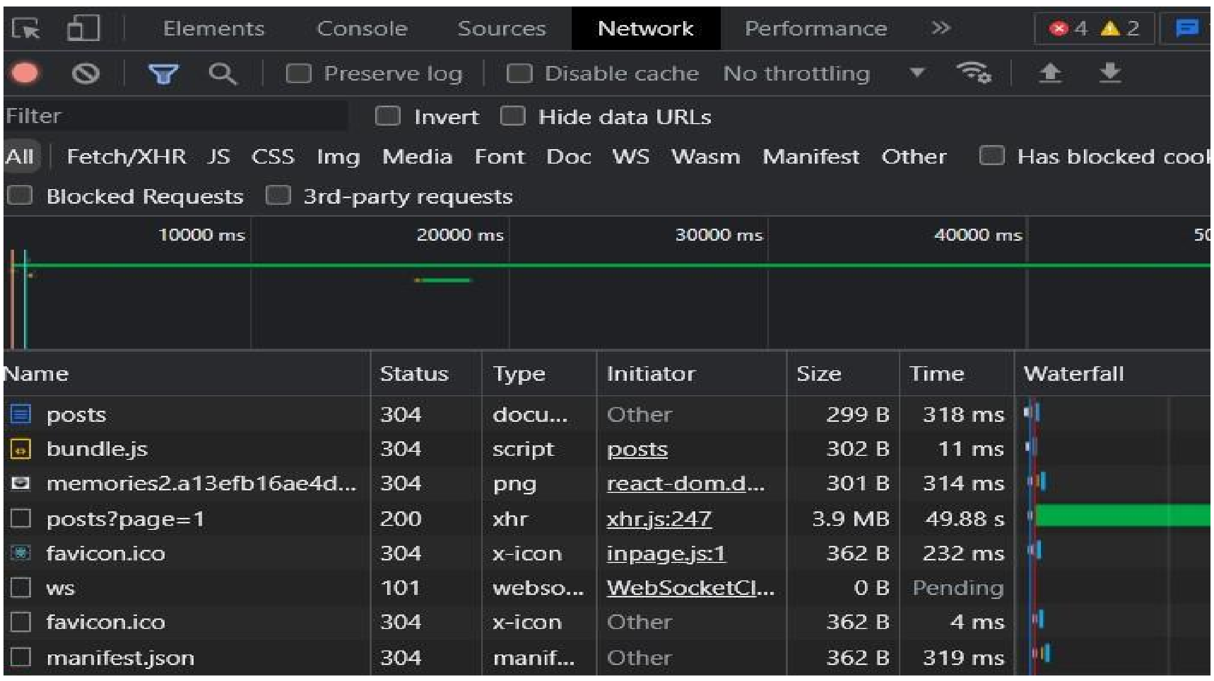
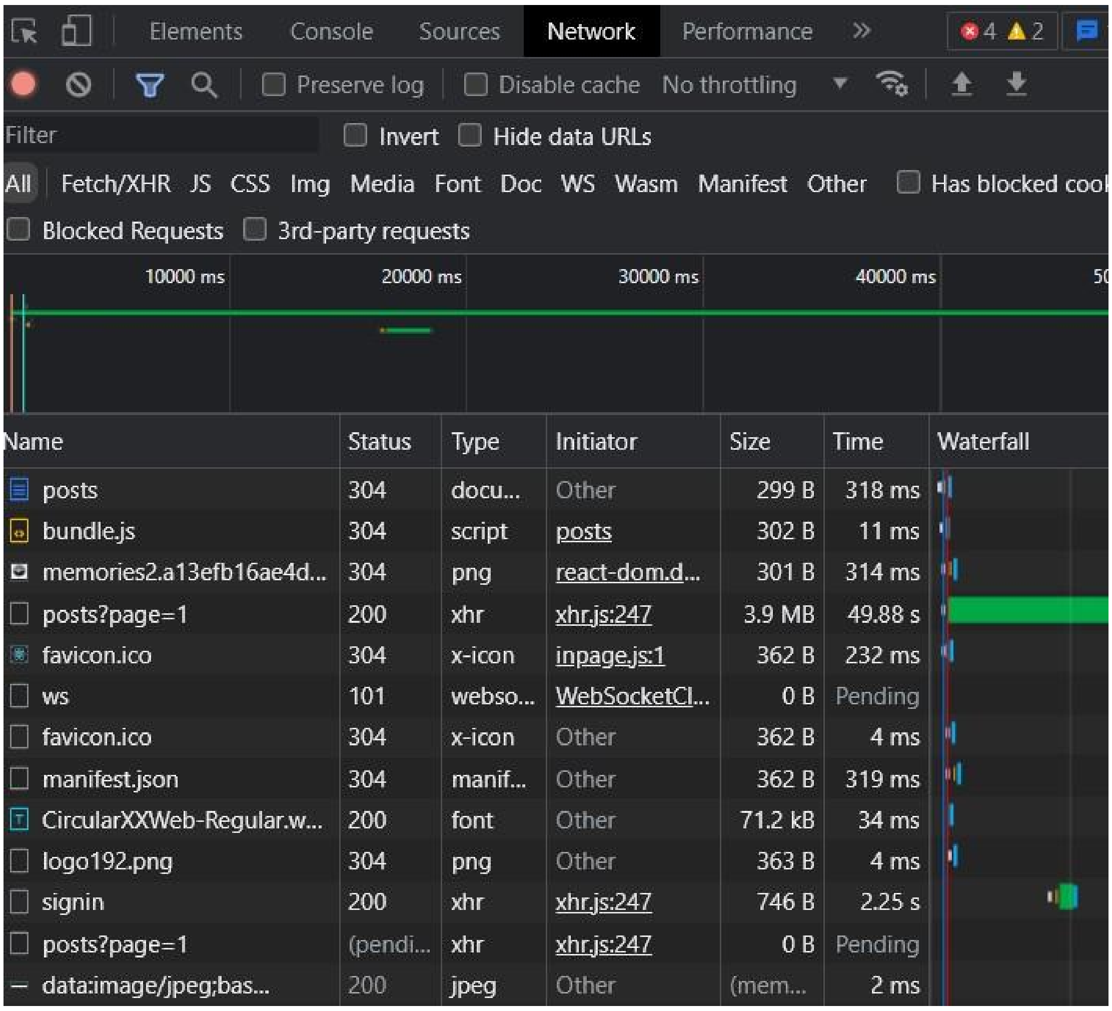


Fig 29 : The register column



Name	Status	Type	Initiator	Size	Time	Waterfall
posts	304	docu...	Other	299 B	318 ms	
bundle.js	304	script	posts	302 B	11 ms	
memories2.a13efb16ae4d...	304	png	react-dom.d...	301 B	314 ms	
posts?page=1	200	xhr	xhr.js:247	3.9 MB	49.88 s	
favicon.ico	304	x-icon	inpage.js:1	362 B	232 ms	
ws	101	webso...	WebSocketCl...	0 B	Pending	
favicon.ico	304	x-icon	Other	362 B	4 ms	
manifest.json	304	manif...	Other	362 B	319 ms	

Fig 30 : Network section before the user is logged in



Name	Status	Type	Initiator	Size	Time	Waterfall
posts	304	docu...	Other	299 B	318 ms	
bundle.js	304	script	posts	302 B	11 ms	
memories2.a13efb16ae4d...	304	png	react-dom.d...	301 B	314 ms	
posts?page=1	200	xhr	xhr.js:247	3.9 MB	49.88 s	
favicon.ico	304	x-icon	inpage.js:1	362 B	232 ms	
ws	101	webso...	WebSocketCl...	0 B	Pending	
favicon.ico	304	x-icon	Other	362 B	4 ms	
manifest.json	304	manif...	Other	362 B	319 ms	
CircularXXWeb-Regular.w...	200	font	Other	71.2 kB	34 ms	
logo192.png	304	png	Other	363 B	4 ms	
signin	200	xhr	xhr.js:247	746 B	2.25 s	
posts?page=1	(pendi...	xhr	xhr.js:247	0 B	Pending	
data:image/jpeg;bas...	200	jpeg	Other	(mem...	2 ms	

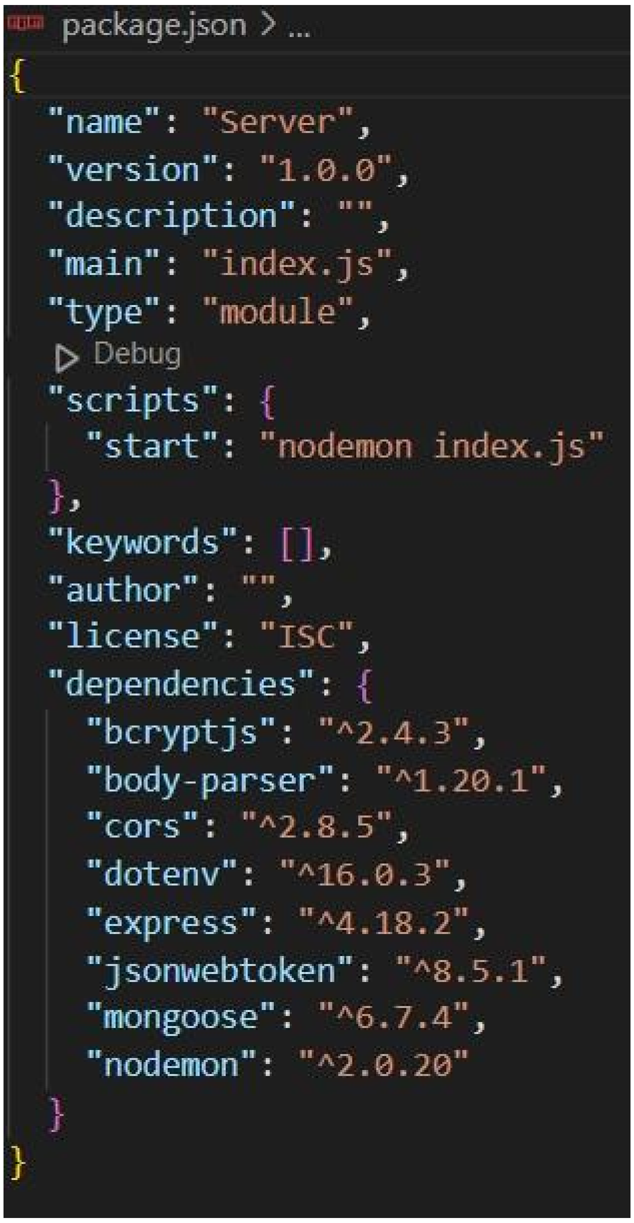
Fig 31 : Network section after the user is logged in

After the user is logged in, we can see that a signin file of ‘xhr’ type appears in the network tab. After that all the posts of page 1 loads. The same xhr type file appeared before the user was signed in. This shows that when the user is logged in, the home page reloads all its contents.

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

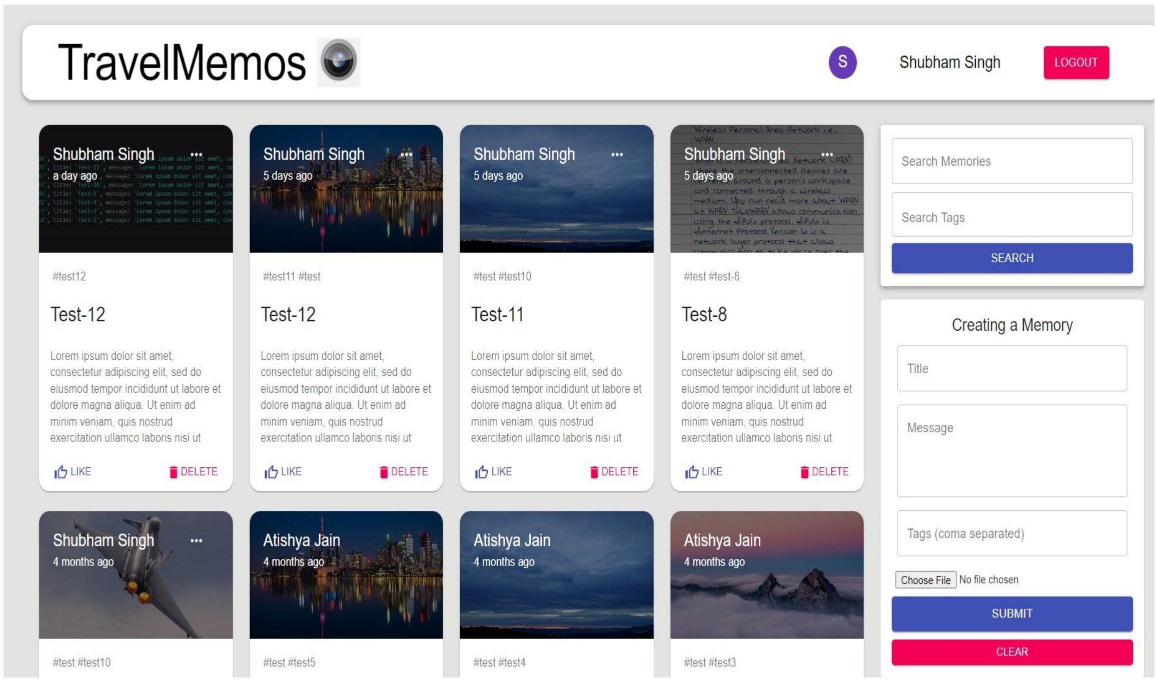


```
package.json > ...
{
  "name": "Server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  ▷ Debug
  "scripts": {
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.20.1",
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "jsonwebtoken": "^8.5.1",
    "mongoose": "^6.7.4",
    "nodemon": "^2.0.20"
  }
}
```

Fig 32 : 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 The Complete UI



Shubham Singh
a day ago
#test12
Test-12
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut

Shubham Singh
5 days ago
#test11 #test
Test-12
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut

Shubham Singh
5 days ago
#test #test10
Test-11
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut

Shubham Singh
5 days ago
#test #test-8
Test-8
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut

Shubham Singh
4 months ago
#test #test10

Atishya Jain
4 months ago
#test #test5

Atishya Jain
4 months ago
#test #test4

Atishya Jain
4 months ago
#test #test3

Fig 33 : The complete ui

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. Through this project, I was able to achieve these goals. Doing internship here 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. The users can click on any post to open it and read the full post.
2. Adding the option of commenting on a particular post.
3. 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.
4. Providing a separate and better 'compose' page.
5. Providing third-party authentication.

References

[1] Sourabh Mahadev Malewade, Archana Ekbote (2021) “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(2022) “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 (2021)“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”(2022), publisher: IJRASET, vol. 10(1), doi : 10.22214/ijraset.2022.39839, pp. 359-366.

SHUBHAM SINGH 191402

ORIGINALITY REPORT

14%	11%	3%	9%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.freecodecamp.org Internet Source	2%
2	Submitted to Coventry University Student Paper	1%
3	stackoverflow.com Internet Source	1%
4	www.theseus.fi Internet Source	1%
5	dev.to Internet Source	1%
6	Submitted to National School of Business Management NSBM, Sri Lanka Student Paper	1%
7	ten-tools.com Internet Source	1%
8	www.geeksforgeeks.org Internet Source	1%
9	glisc.info Internet Source	<1%