BLOG WEBSITE

A PROJECT REPORT
Submitted By

PIYUSH RAJPUT
(University Roll No.2100290140101)
SHIVAM KUMAR
(University Roll No.2100290140128)
VIKAS PARMAR
(University Roll No.2100290140146)

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Mr. Praveen Kumar Gupta
(Assistant Professor)

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206
(JUNE 2023)

DECLARATION

I hereby declare that the work presented in this report entitled “Blog Website”, was carried out by us. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute.
I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully responsible and answerable.

Name: Piysuh Rajput
Roll No: 2100290140101
Branch: Master of Computer Applications

(Candidate Signature)

Name: Shivam Kumar
Roll No: 2100290140128
Branch: Master of Computer Applications

(Candidate Signature)

Name: Vikash Parmar
Roll No.: 2100290140146
Branch: Master of Computer Applications

(Candidate Signature)

CERTIFICATE

Certified that **Piyush Rajput, Shivam Kumar, Vikas Parmar** have carried out the project work having "**BLOG WEBSITE**" for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Technical University, Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself / herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.
Date:

Piyush Rajput(2100290140101)

Shivam Kumar(2100290140128)

Vikas Parmar(2100290140146)

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

Date:

Mr. Praveen Kumar Gupta
(Assistant Professor)
Department of Computer Applications
KIET Group of Institutions, Ghaziabad

Signature of Internal Examiner **Signature of External Examiner**

Dr. Arun Tripathi
Head, Department of Computer Applications
KIET Group of Institutions, Ghaziabad

ABSTRACT

The purpose of Blog Website is to automate the existing manual system by the help of computerized and full-fledged computer software, fulfilling their requirements, so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same. The required software and hardware are easily available and easy to work with.

Online Blogging System, as described above, can lead to error free, secure, reliable and fast management system. It can assist the user to concentrate on their other activities rather to concentrate on the record keeping. Thus, it will help organization in better utilization of resources. The organization can maintain computerized records without redundant entries. That means that one need not be distracted by information that is not relevant, while being able to reach the information.

The aim is to automate its existing manual system by the help of computerized equipment and full-fledged computer software, fulfilling their requirements, so that their valuable data/information can be stored for a longer period with easy accessing and manipulation of the same. Basically, the project describes how to manage for good performance and better services for the clients.

It is also use for creating posting and viewing the others post, ideals, entries and blogs.

ACKNOWLEDGEMENT

Success in life is never attained single-handedly. My deepest gratitude goes to my thesis supervisor, **Mr. Praveen Kumar Gupta** for his guidance, help and encouragement throughout my research work. Their enlightening ideas, comments, and suggestions. Words are not enough to express my gratitude to **Dr. Arun K Tripathi** Professor and Head, Department of Computer Applications, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Piyush Rajput(2100290140101)

Shivam Kumar(2100290140128)

Vikas Parmar(2100290140146)

TABLE OF CONTENT

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Project Description

The "Online Blogging System" has been developed to override the problems prevailing in the practicing manual system. This software is supported to eliminate and, in some cases, reduce the hardships faced by this existing system. Moreover, this system is designed for the need of the company to carry out operations in a smooth and effective manner.

The application is reduced as much as possible to avoid errors while entering the data. It also provides error message while entering invalid data. No formal knowledge is needed for the user to use this system. Thus, by this all it proves it is user-friendly. Online Blogging System, as described above, can lead to error free, secure, reliable and fast management system. It can assist the user to concentrate on their other activities rather to concentrate on the record keeping. Thus, it will help organization in better utilization of resources.

Every organization, whether big or small, has challenges to overcome and managing the information of Comment, Blogs, New Blog, Comment, Technology Blog. Every Online Blogging System has different Blogs needs; therefore, we design exclusive employee management systems that are adapted to your managerial requirements. This is designed to assist in strategic planning and will help you ensure that your organization is equipped with the right level of information and details for your future goals. Also, for those busy executive who are always on the go, our systems come with remote access features, which will allow you to manage your workforce anytime, always. These systems will ultimately allow you to better manage resources.

1.2 Project Scope

The Online Blogging System will allow the users to publish the writings, videos, images or audios if he/she should have credentials to login. The main users of this project are students, teachers and administrators. From an end-user perspective, the Online Blogging System project consists of following functional elements:

1. **Dashboard:** It is the default page of the site and we can access this option from left hand side anytime. All links are available on this page. We can also find the Quick Draft and Activity section here.
2. **Posts**: Here we can see all the published contents by clicking on "All Posts" option and we can also publish new content by "Add New" option.
3. **Media:** We can see the uploaded media items (videos, images, audios) by clicking on "Library" option and we can also add new media item from local system with the help of "Add New" button
4. **Comments:** In this section, we can check that who, when and what has been commented.
5. **Profile:** In this section, we can personalize our profile like Password Change, Profile Picture Change, Display Name, Nickname etc.
6. **Tools:** This option is having additional plugins to install, which may enhance the current functionality.
7. **Collapse:** We can collapse the menu with this option.
8. **Home Button:** It is located at the top left portion and we can use this button to check the timeline where we can view the contents published by everyone.

1.3 Project Category

It is an application that can be accessed over web.

1.4 Motivation and Scope

The list of the idea is to digitalize the process of manually driven ideas, topics and blog sharing among the user, small or big organizations as well as small forums and then to bake it into a web application to make it available for common peopleeven to speed up process reducing further hassle and issues.

It may help collecting perfect management in detail. In a very short time, the collection will be obvious, simple and sensible. It will help a person to know the management of passed year perfectly and vividly. It also helps in current all works relative to Blog website. It will be also reduced the cost of collecting themanagement & collection procedure will go on smoothly.

1.5 Problem Statement

The old manual system was suffering from a series of drawbacks. Since whole of the system was to be maintained with hands the process of keeping, maintaining and retrieving the information was very tedious and lengthy. The records were never used to be in a systematic order. There used to be lots of difficulties in associating any particular transaction with a particular context. There would always be unnecessary consumption of time while entering records and retrieving records. One more problem was that it was very difficult to find errors while entering the records. Once the records were entered it was very difficult to update theserecords.

The reason behind it is that there is a lot of information to be maintained and must be kept in mind while running the business. For these reasons we have provided features Present system is partially automated, existing system is quite laborious as one must enter same information at three different places.

1.6 Existing System

Existing system is manual system. It requires lots of file work to be done. It is a time-consuming system. All user information is maintained manually. Any searching requires so much effort manually. There is no way of spreading the information so fast and in cheapest manner in previous system all information does not get into one place, here people can write whatever they want to write.

1.7 Drawback of existing System

Data redundancy and formatting – the various files are likely to have different formatsand therefore lead to redundancy and inconsistency.

Maintaining registers is costly – traditionally documents have been stored in batches and they filed in file cabinets and boxes. A numerical system is assigned specifically a user number assigned to organize the files.

Error prone – existing system are error prone, since manual work is required. More time is consumed, and errors may propagate due to human mistakes.

Low security feature – due to maintenance of record manually and shared and could view easily by anyone also these could be possible loss of data and confidential information due to some disaster in form of fire, theft etc.

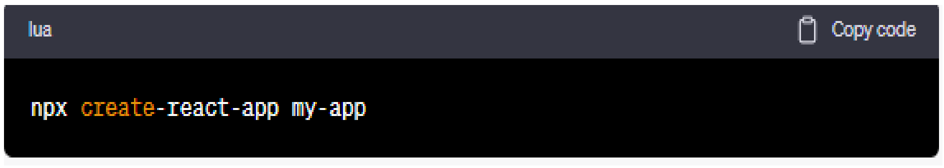
1.8 Virtual environment of React

1.Install Node.js: React requires Node.js, a JavaScript runtime, to build and run applications. Visit the official Node.js website (https://nodejs.org) and download the latest

LTS (Long-Term Support) version for your operating system. Follow the installation instructions for your platform.

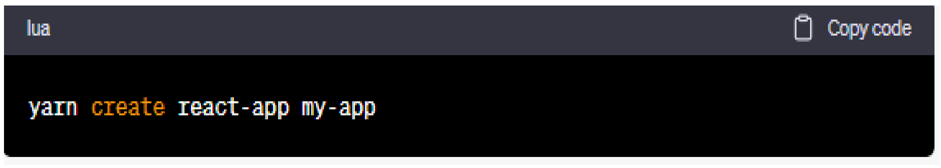
2.Create a new React project: Once Node.js is installed, you can use the Node Package Manager (NPM) or Yarn to create a new React project. Open your command-line interface and run one of the following commands:

Using NPM:1



```
npx create-react-app my-app
```

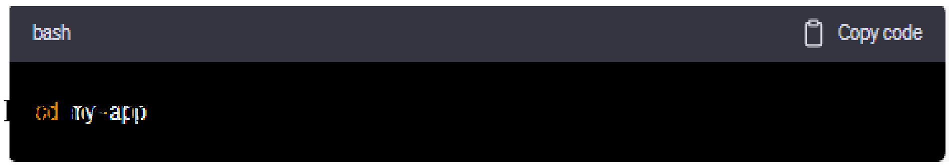
Using Yarn:



```
yarn create react-app my-app
```

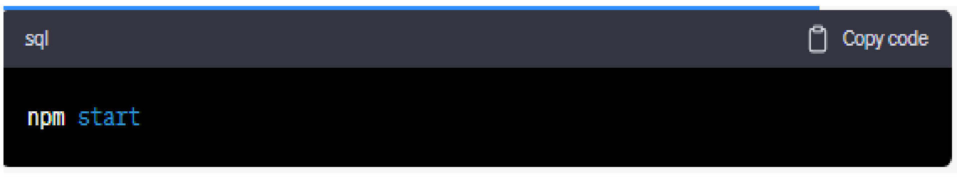
Replace "my-app" with the desired name for your project. This command will generate a new React project with all the necessary files and dependencies.

3.Navigate to the project directory: After the project is created, navigate to the project directory using the following command:



```
cd my-app
```

4.Start the development server: To view and test your React application locally, start the development server. Run the following command within the project directory:



```
npm start
```

This command will start the development server and automatically open your application in a web browser. You should see the default React application template.

Congratulations! You have successfully set up your React development environment. You can now start building your React components and applications.

1.9 Setup of virtual environment of Mongo dB:

A virtual environment is a self-contained directory tree that contains dependencies required by different projects isolated to existing packages.

By using virtual Node.js packages in.Node.js applications

Step 1:

Open your terminal and create a package to store all your virtual environments, using the command "npm init " which is an acronym of "Package-json".

Step 2:

Open your terminal rum a command in npm nodemon, npm express, npm i mongodb to store the package in dependencies

Step 3:

Open your terminal and run npm start. for nodemon

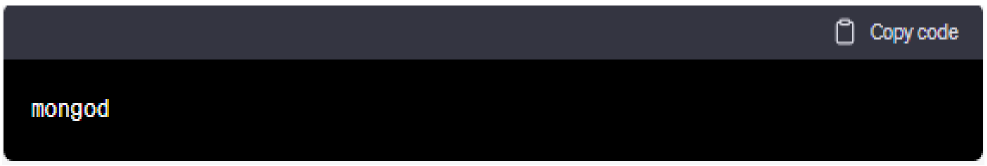
To use MongoDB in a Node.js application, you need to follow these steps to set it up:

1.Install MongoDB:

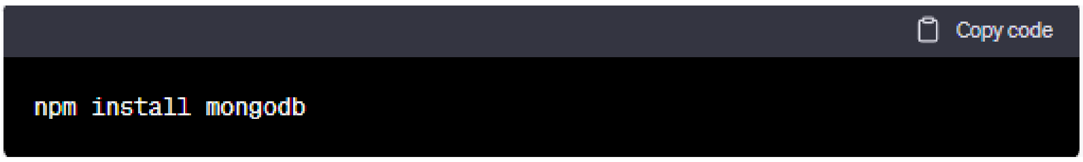
Download and install MongoDB from the official MongoDB website based on your operating system. Follow the installation instructions provided.

2. Set Up a MongoDB Server:

After installing MongoDB, you need to start a MongoDB server to connect your Node.js application to it. Open a terminal or command prompt window and run the following command:



```
mongod
```



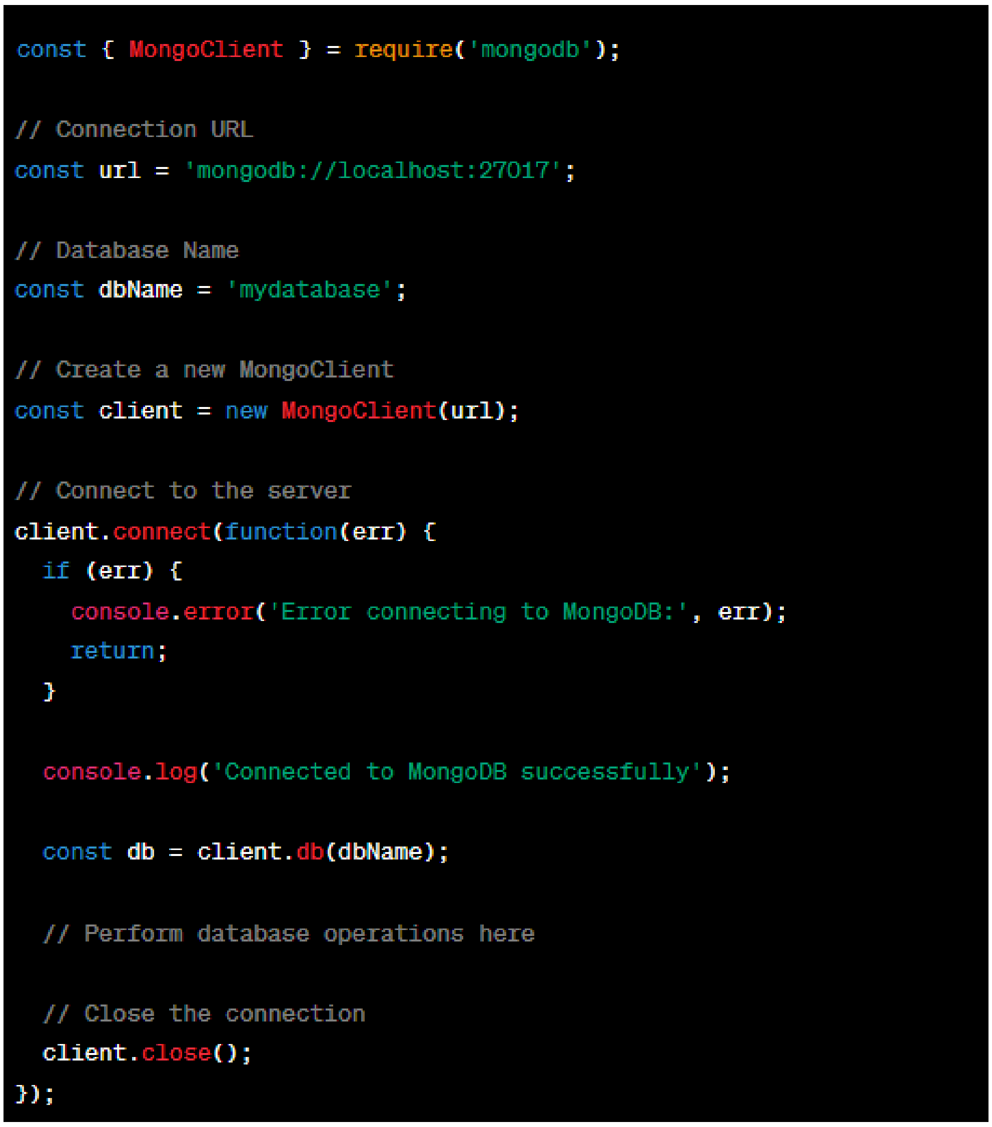
```
npm install mongodb
```

3.Set up a Node.js Project

Create a new directory for your Node.js project and navigate to it in your terminal or command prompt. Run the following command to initialize a new Node.js project:

4.Connect to MongoDB in your Node.js Application

In your Node.js application code, you can use the following code to connect to the MongoDB :



```
const { MongoClient } = require('mongodb');

// Connection URL
const url = 'mongodb://localhost:27017';

// Database Name
const dbName = 'mydatabase';

// Create a new MongoClient
const client = new MongoClient(url);

// Connect to the server
client.connect(function(err) {
  if (err) {
    console.error('Error connecting to MongoDB:', err);
    return;
  }

  console.log('Connected to MongoDB successfully');

  const db = client.db(dbName);

  // Perform database operations here

  // Close the connection
  client.close();
});
```

Modify the url variable if your MongoDB server is running on a different port or host. Also, update the db Name variable with the name of your desired database.

After connecting to the MongoDB server, you can perform various database operations using the db object.

1.10. Perform Database Operations:

Once connected to the MongoDB server, you can perform operations like inserting, updating, querying, and deleting data using the MongoDB driver. Refer to the MongoDB driver documentation for more information on performing different operations.

That's it! You have now set up MongoDB in your Node.js application. Remember to handle errors, close the connection when you're done, and ensure proper error handling and security practices while working with databases.

Chapter 2

Feasibility Study

This project will be developed on computer, so first check whether the technology is technically available or not. Now a day's computer interaction with any job becomes common for any kind of job or work. And because of increasing usage of Computer, Computer is also available with a variety of hardware. Users can fulfil any type of hardware requirement.

Preliminary investigation of a system examines the feasibility of a system that is useful to an organization. It is the first phase of system development.

The main objective of this phase is to identify the current deficiencies in the user's environment and to determine which existing problem are going to be solve in proposed system and which new function needs to be added in proposed system.

An important outcome of such preliminary investigation is to determine whether the system at will meet all needed requirements.

Thus, three tests are carried out on the system namely operation, technical and economical.

2.1 Technical Feasibility

Technical Feasibility examines whether the technology needed is available and if it is available then it feasible to carry out all project activities. The technical needs of a system include:

1 The facility to produce outputs in each time.
2 Ability to process large number of transactions at a particular speed.
3 Giving response to users under certain conditions.

The technology needed for our system is mainly:

1. Latest version of browsers.

2. Any operating system.

These technologies are available which helps to carry out the system efficiently.

2.2 Operational Feasibility

Any project is beneficial if and only satisfies the organization's requirement. For any new system setup, it only meets to be communicated and work the other supporting system.

The new system meets all existing operations since it provides right information at a right time to the right user. A Leigh man can easily operate with the system.

2.3 Behavioural Feasibility

It evaluates and estimates the user attitude or behaviour towards the development of new system. It helps in determining if the system requires special effort to educate, retrain, transfer, and changes in employee's job status on new ways of conducting business

2.4 Economical Feasibility

Economic feasibility of a system examines whether the finance is available for implementing the new system and whether the money spent is recoverable the satisfaction. The cost involves is in designing and developing a good investment for the organization. Thus, hardware requirements used for proposed system are very standard. Moreover, by making use of proposed system to carry out the work speedily will increase and saves the valuable time of an organization.

In the proposed system the finance is highly required for the installation of the software's which can also be recovered by implementing a better system.

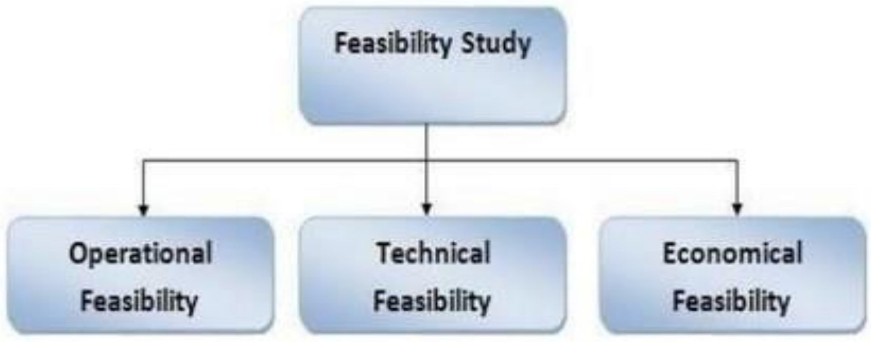


Fig 2.1 Feasibility Study

Chapter 3

Database Design

Process Flow Diagram

A Process Flow Diagram (PFD) is a type of flowchart that illustrates the relationships between major components at an industrial plant. It's most often used in chemical engineering and process engineering, though its concepts are sometimes applied to other processes as well. It’s used to document a process, improve a process or model a new one. Depending on its use and content, it may also be called a Process Flow Chart, Flowsheet, Block Flow Diagram, Schematic Flow Diagram, Macro Flowchart, Top-down Flowchart, Piping and Instrument Diagram, System Flow Diagram or System Diagram. They use a series of symbols and notations to depict a process.

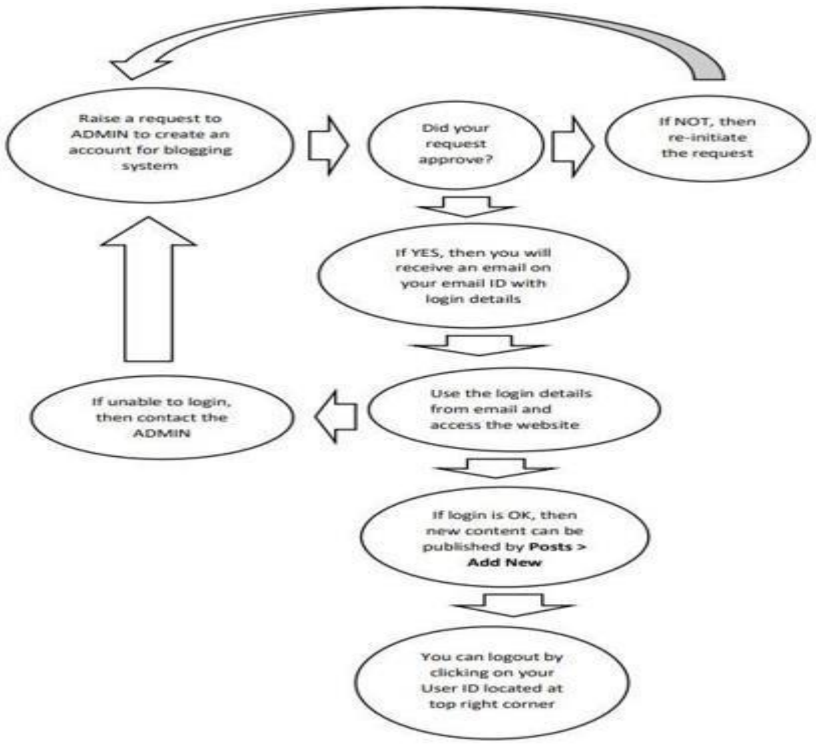

Raise a request to ADMIN to create an account for blogging system
Did your request approve?
If NOT, then re-initiate the request
If YES, then you will receive an email on your email ID with login details
Use the login details from email and access the website
If unable to login, then contact the ADMIN
If login is OK, then new content can be published by Posts > Add New
You can logout by clicking on your User ID located at top right corner

Fig 3.1 Process Flow Diagram

3.2 Data Flow Diagram

A data-flow diagram (DFD) is a graphical representation of the "flow" of data through an information system. DFDs can also be used for the visualization of data processing (structured design). On a DFD, data items flow from an external data source or an internal data store to an internal data store or an external data sink, via an internal process.

3.2.1 Context Level

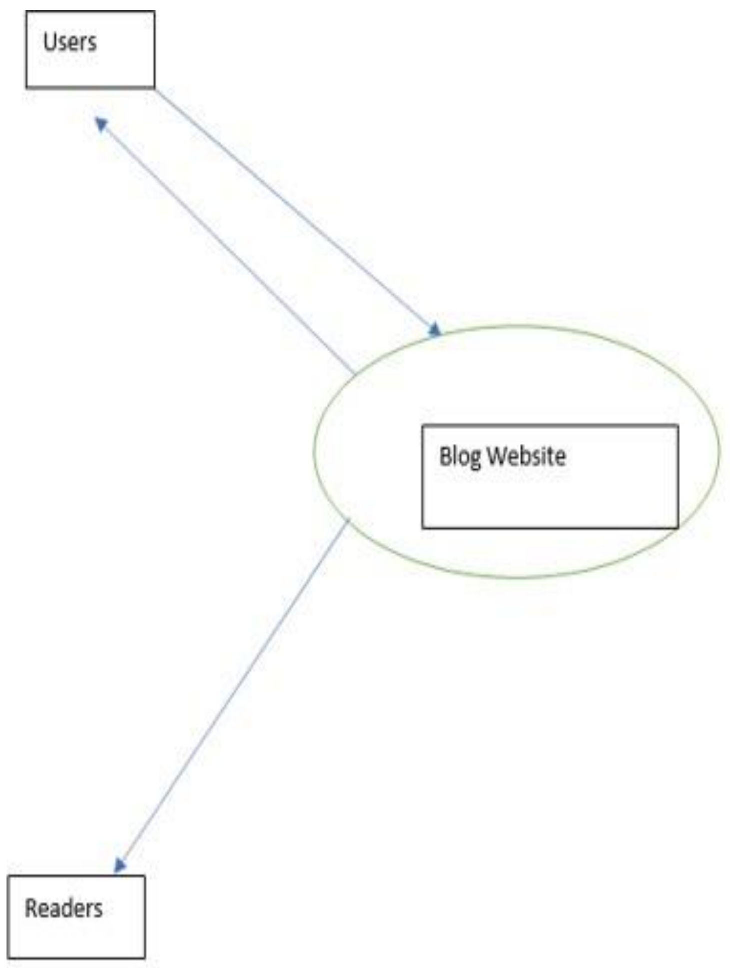


Fig 3.2 Context Level Diagram

This context-level data flow diagram, which shows the interaction between the system and external agents which act as data sources and data sinks. On the context diagram (also known as the Level 0 DFD) the system's interactions with the outside world are modelled purely in terms of data flows across the system boundary. This context diagram shows the entire Online Blogging System as a single process.

3.2.2 High Level Diagram (Level 1)

This level (level 1) shows all processes at the first level of numbering, data stores, external entities and the data flows between them. The purpose of this level is to show the major high-level processes of the Online Blogging System and their interrelation. A level-1 diagram must be balanced with its parent context level diagram, i.e., there must be the same external entities and the same data flows, these can be broken down to more detail in the level 1, e.g., the "Login" data flow could be spilt into "Create and Publish the content" and "Content Removal" and still be valid.

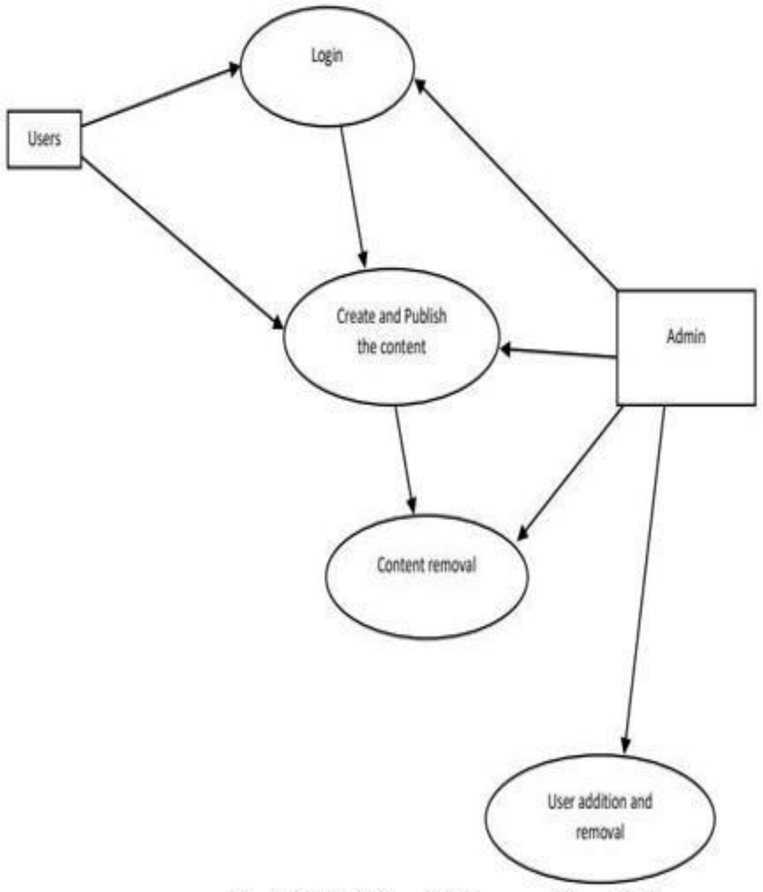


Fig 3.3 High Level Diagram (Level 1)

3.2.3 Use Case Diagram

Use case diagrams are typically developed in the early stage of development and people often apply use case modelling for the following purposes:

1. Specify the context of a system.
2. Capture the requirements of a system.
3. Validate a system architecture.
4. Drive implementation and generate test cases.
5. Developed by analysts together with domain experts.

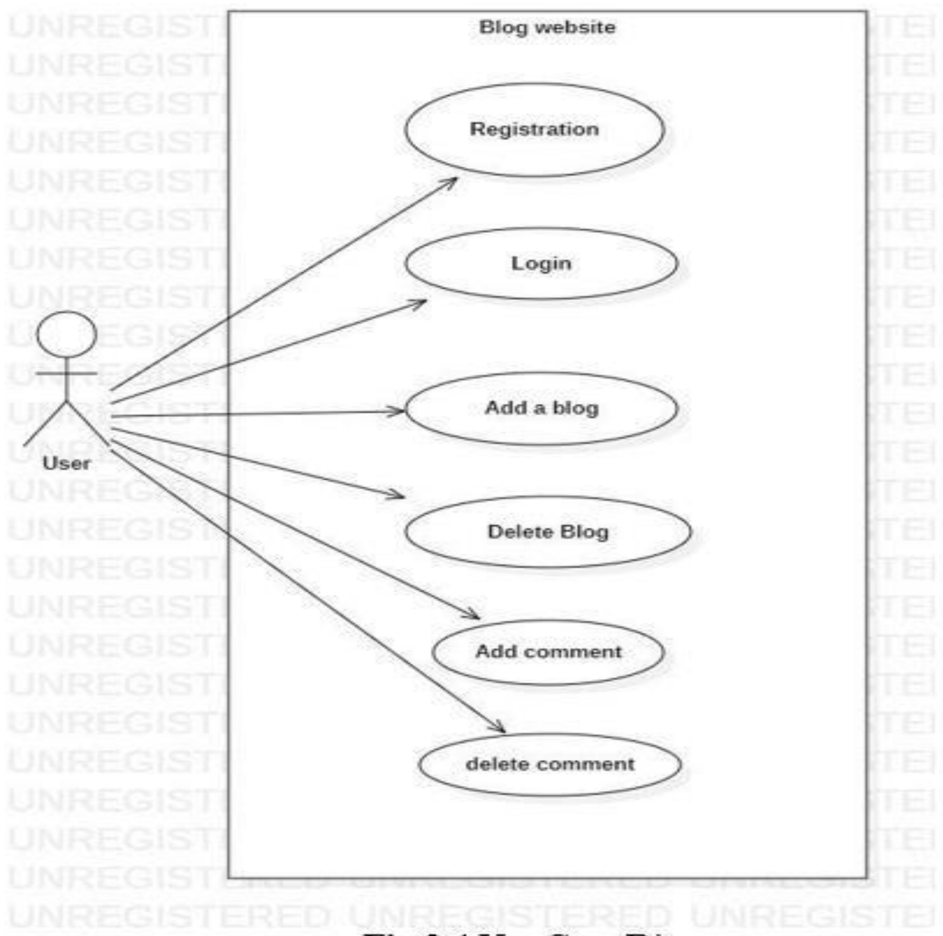


Fig 3.4 Use Case Diagram

Chapter 4

Form Design

4.1 Landing Page

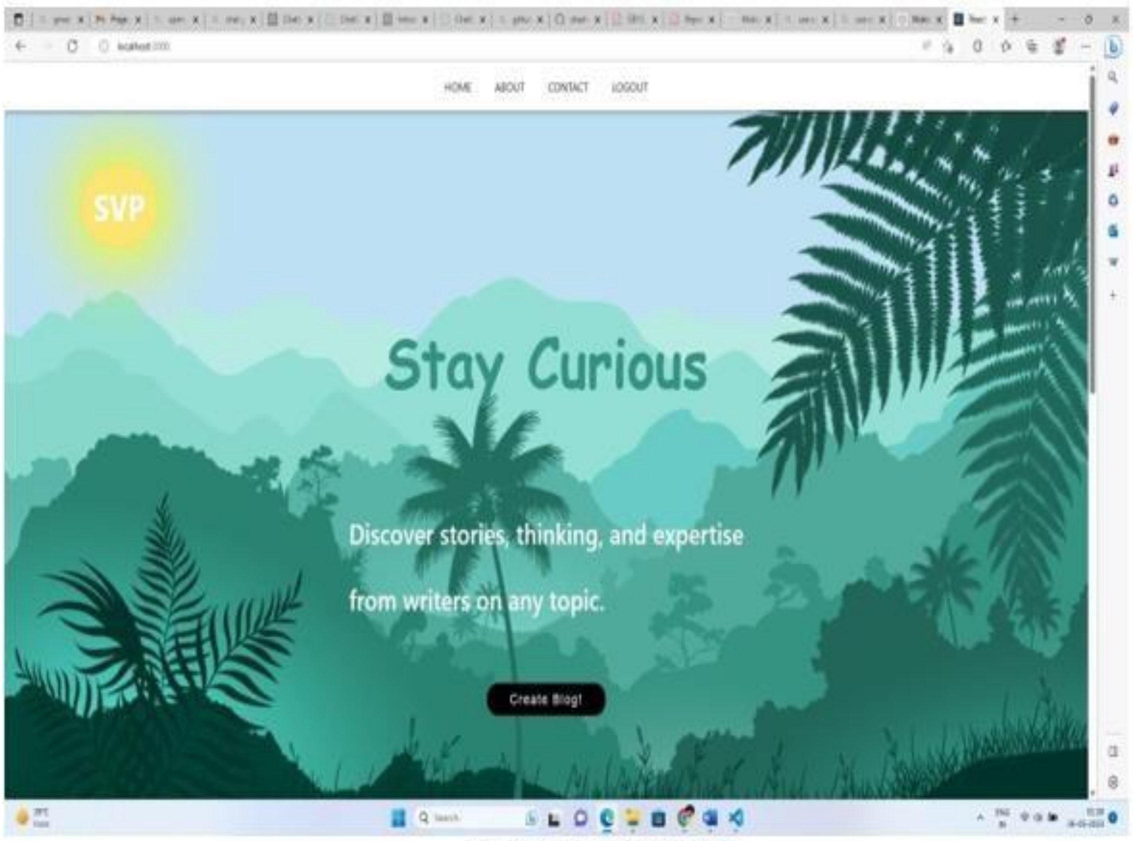
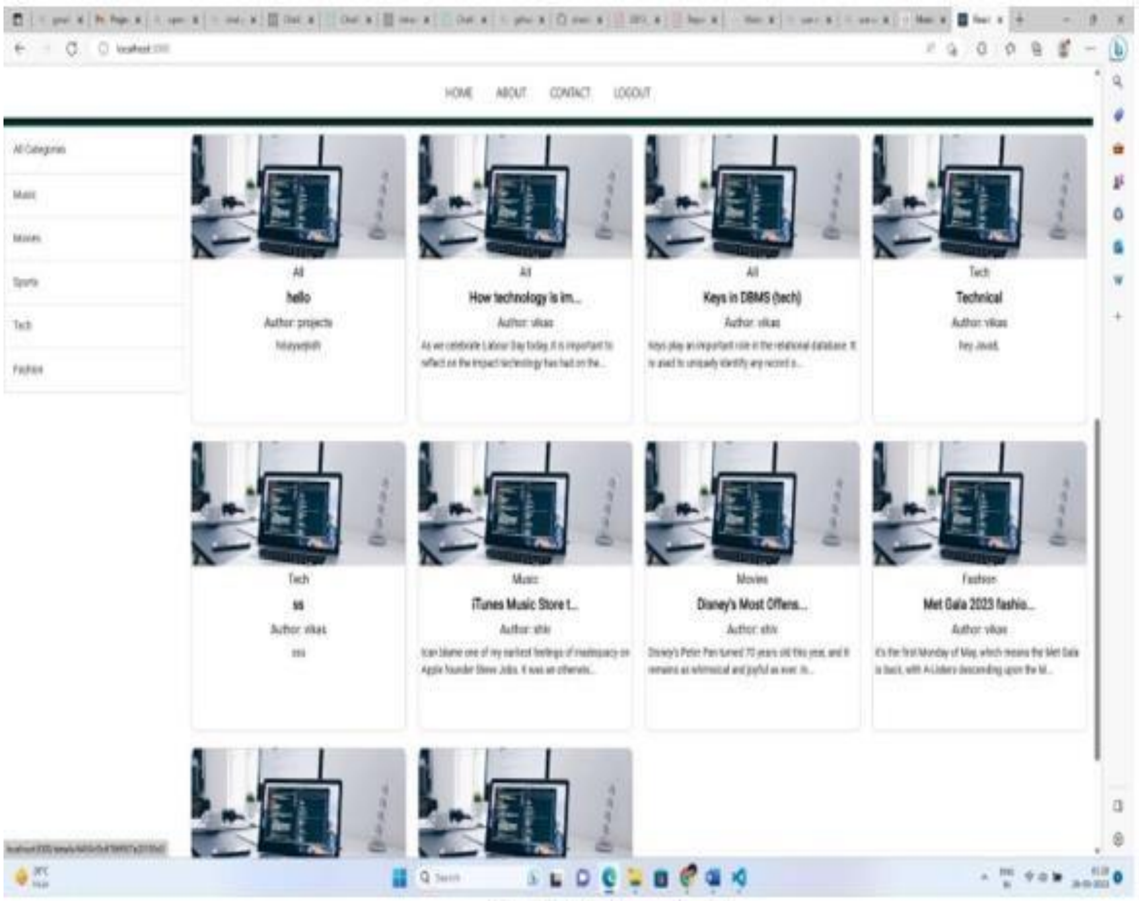

HOME
ABOUT
CONTACT
LOGOUT
SVP
Stay Curious
Discover stories, thinking, and expertise
from writers on any topic.
Create Blog!

Fig 4.1 Landing Page

4.2 Categories Page



All
hello
Author: projects
hiiayuejsdh

All
How technology is im...
Author: vikas
As we celebrate Labour Day today, it is important to reflect on the impact technology has had on the...

All
Keys in DBMS (tech)
Author: vikas
Keys play an important role in the relational database. It is used to uniquely identify any record o...

Tech
Technical
Author: vikas
hey Javed,

Tech
ss
Author: vikas
sss

Music
iTunes Music Store t...
Author: shiv
I can blame one of my earliest feelings of inadequacy on Apple founder Steve Jobs. It was an otherwis...

Movies
Disney's Most Offens...
Author: shiv
Disney's Peter Pan turned 70 years old this year, and it remains as whimsical and joyful as ever. In...

Fashion
Met Gala 2023 fashio...
Author: vikas
It's the first Monday of May, which means the Met Gala is back, with A-Listers descending upon the M...

Fig4.2 Categories Page

4.3 Login Page

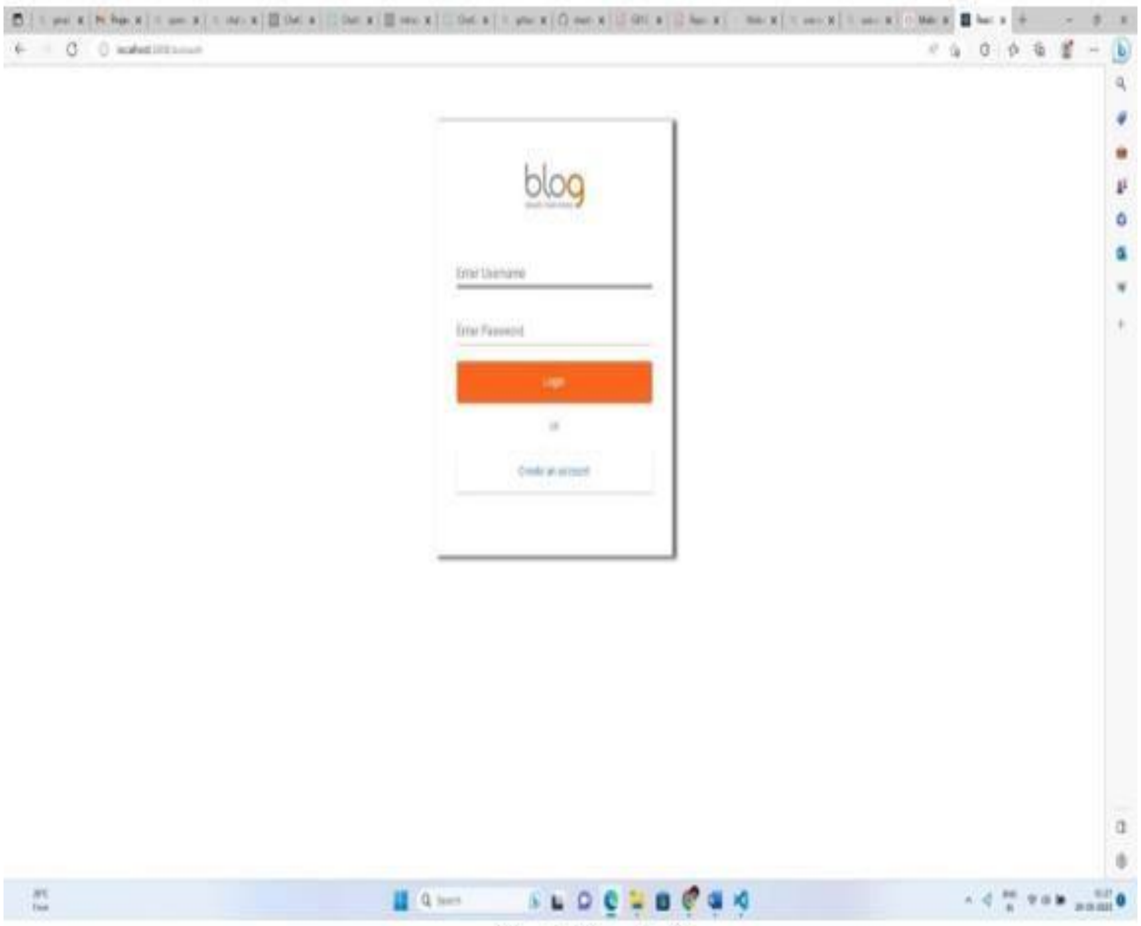


Fig 4.3 Login Page

Chapter 5

Coding Page

5.1 Module Wise

Client Side

Landing.js

```
import React, { useState } from 'react';
import { Link, useSearchParams } from 'react-router-dom';
import '../Landing/Landing.css'
import img1 from '../../imagess/hill1.png';
import img2 from '../../imagess/hill2.png';
import img3 from '../../imagess/hill3.png';
import img4 from '../../imagess/hill4.png';
import img5 from '../../imagess/hill5.png';
import img6 from '../../imagess/leaf.png';
import img7 from '../../imagess/plant.png';
import img8 from '../../imagess/tree.png';
import Logo from '../../../../src/logo.png'


const Landing = () => {
   const [searchParams] = useSearchParams();
   const category = searchParams.get('category');
   const [offset, setOffset] = useState()

   const handleScroll = () => setOffset(window.pageYOffset)

   window.addEventListener('scroll', handleScroll);
   return (<>

      <div className="App">

         <div className="zoom" >
            <header className='outer'>
               <h2 className='logo'>SVP</h2>
```

```
                {/* <img src={Logo} className='logo' alt="" /> */}

            </header>
            <div id='text' style={{ marginTop: (offset * 3.5) + 'px' }}>
                <p className='stay-curious'>Stay Curious</p>
                <p className='discover-stories'>Discover stories, thinking, and expertise
<p>from writers on any topic.</p></p>


                <Link to={`/create?category=${category || ''}`} style={{ textDecoration:
'none' }}>

                    <a id="btn" href="#" ><span>Create Blog!</span></a>
                </Link>


            </div>
            <img src={img1} alt="" id='img1' style={{ top: (1.5 * offset) + 'px' }} />
            {/* <p className='stay-curious'>Stay Curious</p> */}
            <img src={img2} alt="" id='img2' />
            <img src={img3} alt="" id='img3' />
            <img src={img4} alt="" id='img4' style={{ left: (-1.5 * offset) + 'px' }} />
            <img src={img5} alt="" id='img5' style={{ left: (1.5 * offset) + 'px' }} />
            <img src={img6} alt="" id='1mg6' style={{ top: (-1.5 * offset) + 'px' }} />
            <img src={img7} alt="" id='img7' />
            <img src={img8} alt="" id='img8' />

        </div>

    </div>

  </>
  );
}



export default Landing;
```

Posts.jsx

```
import { useEffect, useState } from 'react';

import { Grid, Box } from '@mui/material';
import { Link, useSearchParams } from 'react-router-dom';

// import { getAllPosts } from '../../../service/api';
import { API } from '../../../service/api';

//components
import Post from './Post';

const Posts = () => {
   const [posts, getPosts] = useState([]);

   const [searchParams] = useSearchParams();
   const category = searchParams.get('category');

   useEffect(() => {
      const fetchData = async () => {
         let response = await API.getAllPosts({ category : category || '' });
         if (response.isSuccess) {
            getPosts(response.data);
         }
      }
      fetchData();
   }, [category]);
return (
      <>
         {
            posts?.length ? posts.map(post => (
               <Grid item lg={3} sm={4} xs={12}>
                  <Link style={{textDecoration: 'none', color: 'inherit'}}
to={`details/${post._id}`}>
                     <Post post={post} />
                  </Link>
               </Grid>
            )) : <Box style={{color: '878787', margin: '30px 80px', fontSize: 18}}>
                  No data is available for selected category
               </Box>
         }
      </>
   )
}

export default Posts;
```

Post.jsx

```
import { styled, Box, Typography } from '@mui/material';

const Container = styled(Box)`
   border: 1px solid #d3cede;
   border-radius: 10px;
   margin: 10px;
   display: flex;
   align-items: center;
   flex-direction: column;
   height: 350px;
   & > img, & > p {
      padding: 0 5px 5px 5px;
   }
`;

const Image = styled('img')({
   width: '100%',
   objectFit: 'cover',
   borderRadius: '10px 10px 0 0',
   height: 150
});

const Text = styled(Typography)`
   color: #878787
   font-size: 12px;
`;

const Heading = styled(Typography)`
   font-size: 18px;
   font-weight: 600
`;

const Details = styled(Typography)`
   font-size: 14px;
   word-break: break-word;
`;

const Post = ({ post }) => {
   const url = post.picture ? post.picture : 'https://images.unsplash.com/photo-
1498050108023-
c5249f4df085?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ix
lib=rb-1.2.1&auto=format&fit=crop&w=752&q=80';

   const addEllipsis = (str, limit) => {
      return str.length > limit ? str.substring(0, limit) + '...' : str;
   }
```

```
    return (
        <Container>
            <Image src={url} alt="post" />
            <Text>{post.categories}</Text>
            <Heading>{addEllipsis(post.title, 20)}</Heading>
            <Text>Author: {post.username}</Text>
            <Details>{addEllipsis(post.description, 100)}</Details>
        </Container>
    )
}

export default Post;
```

Categories.jsx

```
import { Button, Table, TableHead, TableRow, TableCell, TableBody, styled } from
'@mui/material';
import { Link, useSearchParams } from 'react-router-dom';

import { categories } from '../../constants/data';

const StyledTable = styled(Table)`
    border: 1px solid rgba(224, 224, 224, 1);
`;

const StyledButton = styled(Button)`
    margin: 20px;
    width: 85%;
    background: #6495ED;
    color: #fff;
    text-decoration: none;
`;

const StyledLink = styled(Link)`
    text-decoration: none;
    color: inherit;
`;

const Categories = () => {
    const [searchParams] = useSearchParams();
    const category = searchParams.get('category');

    return (
        <>
            {/* <Link to={`/create?category=${category || ''}`} style={{ textDecoration:
'none' }}>
```

```
                <StyledButton variant="contained">Create Blog</StyledButton>
            </Link> */}

            <StyledTable>
                <TableHead>
                    <TableRow>
                        <TableCell>
                            <StyledLink to={"/"}>
                                All Categories
                            </StyledLink>
                        </TableCell>
                    </TableRow>
                </TableHead>
                <TableBody>
                    {
                        categories.map(category => (
                            <TableRow key={category.id}>
                                <TableCell>
                                    <StyledLink to={`/?category=${category.type}`}>
                                        {category.type}
                                    </StyledLink>
                                </TableCell>
                            </TableRow>
                        ))
                    }
                </TableBody>
            </StyledTable>
        </>
    )
}

export default Categories;
```

Account

Login.jsx

```
import React, { useState, useEffect, useContext } from 'react';
import { TextField, Box, Button, Typography, styled } from '@mui/material';
import { useNavigate } from 'react-router-dom';

import { API } from '../../service/api';
import { DataContext } from '../../context/DataProvider';

const Component = styled(Box)`
    width: 400px;
    margin: auto;
    box-shadow: 5px 2px 5px 2px rgb(0 0 0/ 0.6);
`;

const Image = styled('img')({
    width: 100,
    display: 'flex',
    margin: 'auto',
    padding: '50px 0 0'
});

const Wrapper = styled(Box)`
    padding: 25px 35px;
    display: flex;
    flex: 1;
    overflow: auto;
    flex-direction: column;
    & > div, & > button, & > p {
        margin-top: 20px;
    }
`;

const LoginButton = styled(Button)`
    text-transform:  none;
    background: #FB641B;
    color: #fff;
    height: 48px;
    border-radius: 2px;
`;

const SignupButton = styled(Button)`
    text-transform: none;
    background: #fff;
    color: #2874f0;
    height: 48px;
    border-radius: 2px;
    box-shadow: 0 2px 4px 0 rgb(0 0 0 / 20%);
`;

const Text = styled(Typography)`
    color: #878787;
    font-size: 12px;
`;

const Error = styled(Typography)`
    font-size: 10px;
    color: #ff6161;
    line-height: 0;
    margin-top: 10px;
    font-weight: 600;
`


const loginInitialValues = {
    username: '',
    password: ''
};

const signupInitialValues = {
    name: '',
    username: '',
    password: '',
};

const Login = ({ isUserAuthenticated }) => {
    const [login, setLogin] = useState(loginInitialValues);
    const [signup, setSignup] = useState(signupInitialValues);
    const [error, showError] = useState('');
    const [account, toggleAccount] = useState('login');

    const navigate = useNavigate();
    const { setAccount } = useContext(DataContext);

    const imageURL = 'https://www.sesta.it/wp-content/uploads/2021/03/logo-blog-
sesta-trasparente.png';

    useEffect(() => {
        showError(false);
    }, [login])

    const onValueChange = (e) => {
        setLogin({ ...login, [e.target.name]: e.target.value });
    }

    const onInputChange = (e) => {
        setSignup({ ...signup, [e.target.name]: e.target.value });
    }

    const loginUser = async () => {
        let response = await API.userLogin(login);
        if (response.isSuccess) {
            showError('');

            sessionStorage.setItem('accessToken', `Bearer ${response.data.accessToken}`);
            sessionStorage.setItem('refreshToken', `Bearer ${response.data.refreshToken}`);
            setAccount({ name: response.data.name, username: response.data.username });

            isUserAuthenticated(true)
            setLogin(loginInitialValues);
            navigate('/');
        } else {
            showError('Something went wrong! please try again later');
        }
    }

    const signupUser = async () => {
        let response = await API.userSignup(signup);
        if (response.isSuccess) {
            showError('');
            setSignup(signupInitialValues);
            toggleAccount('login');
        } else {
            showError('Something went wrong! please try again later');
        }
    }

    const toggleSignup = () => {
        account === 'signup' ? toggleAccount('login') : toggleAccount('signup');
    }

    return (
        <Component>
            <Box>
                <Image src={imageURL} alt="blog" />
                {
                    account === 'login' ?
                        <Wrapper>
                            <TextField variant="standard" value={login.username}
onChange={(e) => onValueChange(e)} name='username' label='Enter Username' />
                            <TextField variant="standard" value={login.password}
onChange={(e) => onValueChange(e)} name='password' label='Enter Password' />

                            {error && <Error>{error}</Error>}
                            <LoginButton variant="contained" onClick={() => loginUser()}
>Login</LoginButton>
                            <Text style={{ textAlign: 'center' }}>OR</Text>
                            <SignupButton onClick={() => toggleSignup()} style={{
marginBottom: 50 }}>Create an account</SignupButton>
                        </Wrapper> :
                        <Wrapper>
                            <TextField variant="standard" onChange={(e) => onInputChange(e)}
name='name' label='Enter Name' />
                            <TextField variant="standard" onChange={(e) => onInputChange(e)}
name='username' label='Enter Username' />
                            <TextField variant="standard" onChange={(e) => onInputChange(e)}
name='password' label='Enter Password' />

                            <SignupButton onClick={() => signupUser()}
>Signup</SignupButton>
                            <Text style={{ textAlign: 'center' }}>OR</Text>
                            <LoginButton variant="contained" onClick={() =>
toggleSignup()}>Already have an account</LoginButton>
                        </Wrapper>
                }
            </Box>
        </Component>
    )
}

export default Login;
```

Create

CreatePost.jsx

```
import React, { useState, useEffect, useContext } from 'react';

import { styled, Box, TextareaAutosize, Button, InputBase, FormControl } from
'@mui/material';
import { AddCircle as Add } from '@mui/icons-material';
import { useNavigate, useLocation } from 'react-router-dom';

import { API } from '../../service/api';
import { DataContext } from '../../context/DataProvider';

const Container = styled(Box)(({ theme }) => ({
    margin: '50px 100px',
    [theme.breakpoints.down('md')]: {
        margin: 0
    }
}));

const Image = styled('img')({
    width: '100%',
    height: '50vh',
    objectFit: 'cover'
});

const StyledFormControl = styled(FormControl)`
    margin-top: 10px;
    display: flex;
    flex-direction: row;
`;

const InputTextField = styled(InputBase)`
    flex: 1;
    margin: 0 30px;
    font-size: 25px;
`;

const Textarea = styled(TextareaAutosize)`
    width: 100%;
    border: none;
    margin-top: 50px;
    font-size: 18px;
    &:focus-visible {
        outline: none;
    }
`;
```

```
const initialPost = {
   title: '',
   description: '',
   picture: '',
   username: '',
   categories: '',
   createdDate: new Date()
}

const CreatePost = () => {
   const navigate = useNavigate();
   const location = useLocation();

   const [post, setPost] = useState(initialPost);
   const [file, setFile] = useState('');
   const { account } = useContext(DataContext);

   const url = post.picture ? post.picture : 'https://images.unsplash.com/photo-
1543128639-
4cb7e6eeef1b?ixid=MnwxMjA3fDB8MHxzZWFyY2h8Mnx8bGFwdG9wJTIwc2V0d
XB8ZW58MHx8MHx8&ixlib=rb-1.2.1&w=1000&q=80';

   useEffect(() => {
      const getImage = async () => {
         if(file) {
            const data = new FormData();
            data.append("name", file.name);
            data.append("file", file);

            const response = await API.uploadFile(data);
            post.picture = response.data;
         }
      }
      getImage();
      post.categories = location.search?.split('=')[1] || 'All';
      post.username = account.username;
   }, [file])

   const savePost = async () => {
      await  API.createPost(post);
      navigate('/');
   }

   const handleChange = (e) => {
      setPost({ ...post, [e.target.name]: e.target.value });
   }

   return (
      <Container>
```

```
            <Image src={url} alt="post" />

            <StyledFormControl>
                <label htmlFor="fileInput">
                    <Add fontSize="large" color="action" />
                </label>
                <input
                    type="file"
                    id="fileInput"
                    style={{ display: "none" }}
                    onChange={(e) => setFile(e.target.files[0])}
                />
                <InputTextField onChange={(e) => handleChange(e)} name='title'
placeholder="Title" />
                <Button onClick={() => savePost()} variant="contained"
color="primary">Publish</Button>
            </StyledFormControl>

            <Textarea
                rowsMin={5}
                placeholder="Tell your story..."
                name='description'
                onChange={(e) => handleChange(e)}
            />
        </Container>
    )
}

export default CreatePost;
```

Update.jsx

```
import React, { useState, useEffect } from 'react';

import { Box, styled, TextareaAutosize, Button, FormControl, InputBase } from
'@mui/material';
import { AddCircle as Add } from '@mui/icons-material';
import { useNavigate, useParams } from 'react-router-dom';

import { API } from '../../service/api';

const Container = styled(Box)(({ theme }) => ({
    margin: '50px 100px',
    [theme.breakpoints.down('md')]: {
        margin: 0
    }
}));

const Image = styled('img')({
    width: '100%',
    height: '50vh',
    objectFit: 'cover'
});

const StyledFormControl = styled(FormControl)`
    margin-top: 10px;
    display: flex;
    flex-direction: row;
`;

const InputTextField = styled(InputBase)`
    flex: 1;
    margin: 0 30px;
    font-size: 25px;
`;

const StyledTextArea = styled(TextareaAutosize)`
    width: 100%;
    border: none;
    margin-top: 50px;
    font-size: 18px;
    &:focus-visible {
        outline: none;
    }
`;

const initialPost = {
    title: '',
    description: '',
    picture: '',
    username: 'codeforinterview',
    categories: 'Tech',
    createdDate: new Date()
}

useEffect(() => {
        const getImage = async () => {
            if(file) {
                const data = new FormData();
                data.append("name", file.name);
                data.append("file", file);

                const response = await API.uploadFile(data);
                post.picture = response.data;
            }
        }
        getImage();
        post.categories = location.search?.split('=')[1] || 'All';
        post.username = account.username;
    }, [file])

    const savePost = async () => {
        await  API.createPost(post);
        navigate('/');
    }

    const handleChange = (e) => {
        setPost({ ...post, [e.target.name]: e.target.value });
    }

const Update = () => {
    const navigate = useNavigate();
    const [post, setPost] = useState(initialPost);
    const [file, setFile] = useState('');
    const [imageURL, setImageURL] = useState('');
    const { id } = useParams();

    const url = 'https://images.unsplash.com/photo-1543128639-
4cb7e6eeef1b?ixid=MnwxMjA3fDB8MHxzZWFyY2h8Mnx8bGFwdG9wJTIwc2V0d
XB8ZW58MHx8MHx8&ixlib=rb-1.2.1&w=1000&q=80';

    useEffect(() => {
        const fetchData = async () => {
            let response = await API.getPostById(id);
            if (response.isSuccess) {
                setPost(response.data);
            }
        }
        fetchData();
    }, []);

    useEffect(() => {
        const getImage = async () => {
            if(file) {
                const data = new FormData();
                data.append("name", file.name);
                data.append("file", file);

                const response = await API.uploadFile(data);
                if (response.isSuccess) {
                    post.picture = response.data;
                    setImageURL(response.data);
                }
            }
        }
        getImage();
    }, [file])

    const updateBlogPost = async () => {
        await API.updatePost(post);
        navigate(`/details/${id}`);
    }

    const handleChange = (e) => {
        setPost({ ...post, [e.target.name]: e.target.value });
    }

    return (
        <Container>
            <Image src={post.picture || url} alt="post" />

            <StyledFormControl>
                <label htmlFor="fileInput">
                    <Add fontSize="large" color="action" />
                </label>
                <input
                    type="file"
                    id="fileInput"
                    style={{ display: "none" }}
                    onChange={(e) => setFile(e.target.files[0])}
                />
                <InputTextField onChange={(e) => handleChange(e)} value={post.title}
name='title' placeholder="Title" />
                <Button onClick={() => updateBlogPost()} variant="contained"
color="primary">Update</Button>
```

```
            </StyledFormControl>

            <StyledTextArea
              rowsMin={5}
              placeholder="Tell your story..."
              name='description'
              onChange={(e) => handleChange(e)}
              value={post.description}
            />
        </Container>
    )
}

export default Update;
```

Constant

Config.js

```
// API NOTIFICATION MESSAGES
export const API_NOTIFICATION_MESSAGES = {
   loading: {
      title: "Loading...",
      message: "Data is being loaded. Please wait"
   },
   success: {
      title: "Success",
      message: "Data successfully loaded"
   },
   requestFailure: {
      title: "Error!",
      message: "An error occur while parsing request data"
   },
   responseFailure: {
      title: "Error!",
      message: "An error occur while fetching response from server. Please try again"
   },
   networkError: {
      title: "Error!",
      message: "Unable to connect to the server. Please check internet connectivity and
try again."
   }
}

// API SERVICE URL
// SAMPLE REQUEST
// NEED SERVICE CALL: { url: "/", method: "POST/GET/PUT/DELETE" }
export const SERVICE_URLS = {
   userLogin: { url: '/login', method: 'POST' },
   userSignup: { url: '/signup', method: 'POST' },
   getAllPosts: { url: '/posts', method: 'GET', params: true },
   getRefreshToken: { url: '/token', method: 'POST' },
   uploadFile: { url: 'file/upload', method: 'POST' },
   createPost: { url: 'create', method: 'POST' },
   deletePost: { url: 'delete', method: 'DELETE', query: true },
   getPostById: { url: 'post', method: 'GET', query: true },
   newComment: { url: '/comment/new', method: 'POST' },
   getAllComments: { url: 'comments', method: 'GET', query: true },
   deleteComment: { url: 'comment/delete', method: 'DELETE', query: true },
   updatePost: { url: 'update', method: 'PUT', query: true }
}
```

Data.js

```
export const categories = [
   { id: 1, type: "Music" },
   { id: 2, type: "Movies" },
   { id: 3, type: "Sports" },
   { id: 4, type: "Tech" },
   { id: 5, type: "Fashion" }
];
```

Context

DataProvide.js

```
import { createContext, useState } from "react";


export const DataContext = createContext(null);

const DataProvider = ({ children }) => {
    const [ account, setAccount ] = useState({ name: '', username: '' });

    return (
        <DataContext.Provider value={{
            account,
            setAccount
        }}>
            {children}
        </DataContext.Provider>
    )
}

export default DataProvider;
```

Service

Api.js

```
import axios from 'axios';

import { API_NOTIFICATION_MESSAGES, SERVICE_URLS } from
'../constants/config';
import { getAccessToken, getRefreshToken, setAccessToken, getType } from
'../utils/common-utils';

const API_URL = 'http://localhost:8000';

const axiosInstance = axios.create({
    baseURL: API_URL,
    timeout: 10000,
    headers: {
        "content-type": "application/json"
    }
});

axiosInstance.interceptors.request.use(
    function(config) {
        if (config.TYPE.params) {
            config.params = config.TYPE.params
        } else if (config.TYPE.query) {
            config.url = config.url + '/' + config.TYPE.query;
        }
        return config;
    },
    function(error) {
        return Promise.reject(error);
    }
);

axiosInstance.interceptors.response.use(
    function(response) {
        // Stop global loader here
        return processResponse(response);
    },
    function(error) {
        // Stop global loader here
        return Promise.reject(ProcessError(error));
    }
)

///////////////////////////////
// If success -> returns { isSuccess: true, data: object }
// If fail -> returns { isFailure: true, status: string, msg: string, code: int }
```

```
////////////////////////////
const processResponse = (response) => {
   if (response?.status === 200) {
      return { isSuccess: true, data: response.data }
   } else {
      return {
         isFailure: true,
         status: response?.status,
         msg: response?.msg,
         code: response?.code
      }
   }
}

/////////////////////////////
// If success -> returns { isSuccess: true, data: object }
// If fail -> returns { isError: true, status: string, msg: string, code: int }
////////////////////////////
const ProcessError = async (error) => {
   if (error.response) {
      // Request made and server responded with a status code
      // that falls out of the range of 2xx
      if (error.response?.status === 403) {
         const { url, config } = error.response;
         console.log(error);
         try {
            let response = await API.getRefreshToken({ token: getRefreshToken() });
            if (response.isSuccess) {
               sessionStorage.clear();
               setAccessToken(response.data.accessToken);

               const requestData = error.toJSON();

               let response1 = await axios({
                  method: requestData.config.method,
                  url: requestData.config.baseURL + requestData.config.url,
                  headers: { "content-type": "application/json", "authorization":
getAccessToken() },
                  params: requestData.config.params
               });
            }
         } catch (error) {
            return Promise.reject(error)
         }
      } else {
         console.log("ERROR IN RESPONSE: ", error.toJSON());
         return {
            isError: true,
            msg: API_NOTIFICATION_MESSAGES.responseFailure,
                code: error.response.status
            }
        }
    } else if (error.request) {
        // The request was made but no response was received
        console.log("ERROR IN RESPONSE: ", error.toJSON());
        return {
            isError: true,
            msg: API_NOTIFICATION_MESSAGES.requestFailure,
            code: ""
        }
    } else {
        // Something happened in setting up the request that triggered an Error
        console.log("ERROR IN RESPONSE: ", error.toJSON());
        return {
            isError: true,
            msg: API_NOTIFICATION_MESSAGES.networkError,
            code: ""
        }
    }
}

const API = {};

for (const [key, value] of Object.entries(SERVICE_URLS)) {
    API[key] = (body, showUploadProgress, showDownloadProgress) =>
        axiosInstance({
            method: value.method,
            url: value.url,
            data: value.method === 'DELETE' ? '' : body,
            responseType: value.responseType,
            headers: {
                authorization: getAccessToken(),
            },
            TYPE: getType(value, body),
            onUploadProgress: function(progressEvent) {
                if (showUploadProgress) {
                    let percentCompleted = Math.round((progressEvent.loaded * 100) /
progressEvent.total);
                    showUploadProgress(percentCompleted);
                }
            },
            onDownloadProgress: function(progressEvent) {
                if (showDownloadProgress) {
                    let percentCompleted = Math.round((progressEvent.loaded * 100) /
progressEvent.total);
                    showDownloadProgress(percentCompleted);
if (error.request) {
        // The request was made but no response was received
        console.log("ERROR IN RESPONSE: ", error.toJSON());
        return {
            isError: true,
            msg: API_NOTIFICATION_MESSAGES.requestFailure,
            code: ""
        }
    }







else {
        // Something happened in setting up the request that triggered an Error
        console.log("ERROR IN RESPONSE: ", error.toJSON());
        return {
            isError: true,
            msg: API_NOTIFICATION_MESSAGES.networkError,
            code: ""
        }
else if (error.request) {
        // The request was made but no response was received
        console.log("ERROR IN RESPONSE: ", error.toJSON());
        return {
            isError: true,
            msg: API_NOTIFICATION_MESSAGES.requestFailure,
            code: ""
        }


                }
            }
        });
}

export { API };
```

Component

about

About.jsx

```
import { Box, styled, Typography, Link } from '@mui/material';
import { GitHub, Instagram, Email } from '@mui/icons-material';

const Banner = styled(Box)`
    background-image: url(https://www.wallpapertip.com/wmimgs/23-236943_us-
wallpaper-for-website.jpg);
    width: 100%;
    height: 50vh;
    background-position: left 0px bottom 0px;
    background-size: cover;
    background-image: url(https://www.wallpapertip.com/wmimgs/23-236943_us-
wallpaper-for-website.jpg);
    width: 100%;
    height: 50vh;
    background-position: left 0px bottom 0px;
    background-size: cover;
    background-image: url(https://www.wallpapertip.com/wmimgs/23-236943_us-
wallpaper-for-website.jpg);
    width: 100%;
    height: 50vh;
    background-position: left 0px bottom 0px;
    background-size: cover;

`;

const Wrapper = styled(Box)`
    padding: 20px;
    & > h3, & > h5 {
        margin-top: 50px;
    }
`;

const Text = styled(Typography)`
    color: #878787;
`;

const About = () => {

    return (
        <Box>
            <Banner/>
            <Wrapper>
                <Typography variant="h3">This is a Simple Blog Website</Typography>
            <Text variant="h5">we are a MCA Student From Kiet group of institute And
                This is Our Group Project
              .<br />

              <Box component="span" style={{ marginLeft: 5 }}>

              </Box>
            </Text>
            <Text variant="h5">
              Reach to us....
              <Box component="span" style={{ marginLeft: 5 }}>


              </Box>
                or send me an Email

            </Text>
          </Wrapper>
        </Box>
    )
}

export default About;
```

account

Login.jsx

```
import React, { useState, useEffect, useContext } from 'react';

import { TextField, Box, Button, Typography, styled } from '@mui/material';
import { useNavigate } from 'react-router-dom';

import { API } from '../../service/api';
import { DataContext } from '../../context/DataProvider';

const Component = styled(Box)`
    width: 400px;
    margin: auto;
    box-shadow: 5px 2px 5px 2px rgb(0 0 0/ 0.6);
`;

const Image = styled('img')({
    width: 100,
    display: 'flex',
    margin: 'auto',
    padding: '50px 0 0'
});

const Wrapper = styled(Box)`
    padding: 25px 35px;
    display: flex;
    flex: 1;
    overflow: auto;
    flex-direction: column;
    & > div, & > button, & > p {
        margin-top: 20px;
    }
`;

const LoginButton = styled(Button)`
    text-transform:  none;
    background: #FB641B;
    color: #fff;
    height: 48px;
    border-radius: 2px;
`;
const Wrapper = styled(Box)`
    padding: 25px 35px;
    display: flex;
    flex: 1;
    overflow: auto;
    flex-direction: column;
    & > div, & > button, & > p {
        margin-top: 20px;
    }


const SignupButton = styled(Button)`
    text-transform: none;
    background: #fff;
    color: #2874f0;
    height: 48px;
    border-radius: 2px;
    box-shadow: 0 2px 4px 0 rgb(0 0 0 / 20%);
`;

const Text = styled(Typography)`
    color: #878787;
    font-size: 12px;
`;

const Error = styled(Typography)`
    font-size: 10px;
    color: #ff6161;
    line-height: 0;
    margin-top: 10px;
    font-weight: 600;
`


const loginInitialValues = {
    username: '',
    password: ''
};

const signupInitialValues = {
    name: '',
    username: '',
    password: '',
};

const Login = ({ isUserAuthenticated }) => {
    const [login, setLogin] = useState(loginInitialValues);
    const [signup, setSignup] = useState(signupInitialValues);
    const [error, showError] = useState('');
    const [account, toggleAccount] = useState('login');

    const navigate = useNavigate();
    const { setAccount } = useContext(DataContext);
    const imageURL = 'https://www.sesta.it/wp-content/uploads/2021/03/logo-blog-
sesta-trasparente.png';

    useEffect(() => {
        showError(false);
    }, [login])

    const onValueChange = (e) => {
        setLogin({ ...login, [e.target.name]: e.target.value });
    }

    const onInputChange = (e) => {
        setSignup({ ...signup, [e.target.name]: e.target.value });
    }

    const loginUser = async () => {
        let response = await API.userLogin(login);
        if (response.isSuccess) {
            showError('');

            sessionStorage.setItem('accessToken', `Bearer ${response.data.accessToken}`);
            sessionStorage.setItem('refreshToken', `Bearer ${response.data.refreshToken}`);
            setAccount({ name: response.data.name, username: response.data.username });

            isUserAuthenticated(true)
            setLogin(loginInitialValues);
            navigate('/');
        } else {
            showError('Something went wrong! please try again later');
        }
    }

    const signupUser = async () => {
        let response = await API.userSignup(signup);
        if (response.isSuccess) {
            showError('');
            setSignup(signupInitialValues);
            toggleAccount('login');
        } else {
            showError('Something went wrong! please try again later');
        }
    }

     const toggleSignup = () => {
        account === 'signup' ? toggleAccount('login') : toggleAccount('signup');
    }

    return (
        <Component>
```

```
            <Box>
                <Image src={imageURL} alt="blog" />
                {
                    account === 'login' ?
                        <Wrapper>
                            <TextField variant="standard" value={login.username}
onChange={(e) => onValueChange(e)} name='username' label='Enter Username' />
                            <TextField variant="standard" value={login.password}
onChange={(e) => onValueChange(e)} name='password' label='Enter Password' />

                            {error && <Error>{error}</Error>}

                            <LoginButton variant="contained" onClick={() => loginUser()}
>Login</LoginButton>
                            <Text style={{ textAlign: 'center' }}>OR</Text>
                            <SignupButton onClick={() => toggleSignup()} style={{
marginBottom: 50 }}>Create an account</SignupButton>
                        </Wrapper> :
                        <Wrapper>
                            <TextField variant="standard" onChange={(e) => onInputChange(e)}
name='name' label='Enter Name' />
                            <TextField variant="standard" onChange={(e) => onInputChange(e)}
name='username' label='Enter Username' />
                            <TextField variant="standard" onChange={(e) => onInputChange(e)}
name='password' label='Enter Password' />

                            <SignupButton onClick={() => signupUser()}
>Signup</SignupButton>
                            <Text style={{ textAlign: 'center' }}>OR</Text>
                            <LoginButton variant="contained" onClick={() =>
toggleSignup()}>Already have an account</LoginButton>
                        </Wrapper>
                }
            </Box>
        </Component>
    )
}

export default Login;
```

banner

Banner.jsx

```
const Wrapper = styled(Box)`
   padding: 25px 35px;
   display: flex;
   flex: 1;
   overflow: auto;
   flex-direction: column;
   & > div, & > button, & > p {
      margin-top: 20px;
   }


import { styled, Box, Typography } from '@mui/material';

const Image = styled(Box)`
   width: 100%;
   background: url(https://images.pexels.com/photos/1714208/pexels-photo-
1714208.jpeg) center/55% repeat-x #000;
   height: 50vh;
   display: flex;
   flex-direction: column;
   align-items: center;
   justify-content: center;
`;

const Heading = styled(Typography)`
   font-size: 70px;
   color: #FFFFFF;
   line-height: 1
`;

const SubHeading = styled(Typography)`
   font-size: 20px;
   background: #FFFFFF;
`;

const Banner = () => {

   return (
      <Image>
         <Heading>BLOG</Heading>
         <SubHeading>Code for Interview</SubHeading>
{
               account === 'login' ?
                  <Wrapper>
```

```
                        <TextField variant="standard" value={login.username}
onChange={(e) => onValueChange(e)} name='username' label='Enter Username' />
                        <TextField variant="standard" value={login.password}
onChange={(e) => onValueChange(e)} name='password' label='Enter Password' />

                        {error && <Error>{error}</Error>}

                        <LoginButton variant="contained" onClick={() => loginUser()}
>Login</LoginButton>
                        <Text style={{ textAlign: 'center' }}>OR</Text>
                        <SignupButton onClick={() => toggleSignup()} style={{
marginBottom: 50 }}>Create an account</SignupButton>
                    </Wrapper> :
                    <Wrapper>
                        <TextField variant="standard" onChange={(e) => onInputChange(e)}
name='name' label='Enter Name' />
                        <TextField variant="standard" onChange={(e) => onInputChange(e)}
name='username' label='Enter Username' />
                        <TextField variant="standard" onChange={(e) => onInputChange(e)}
name='password' label='Enter Password' />

                        <SignupButton onClick={() => signupUser()}
>Signup</SignupButton>
                        <Text style={{ textAlign: 'center' }}>OR</Text>
                        <LoginButton variant="contained" onClick={() =>
toggleSignup()}>Already have an account</LoginButton>

        </Image>
    )
}

export default Banner;
```

Contact

Contact.jsx

```
const Wrapper = styled(Box)`
    padding: 25px 35px;
    display: flex;
    flex: 1;
    overflow: auto;
    flex-direction: column;
    & > div, & > button, & > p {
        margin-top: 20px;
    }

import { Box, styled, Typography, Link } from '@mui/material';
import { GitHub, Instagram, Email } from '@mui/icons-material';

const Banner = styled(Box)`
    background-image: url(http://mrtaba.ir/image/bg2.jpg);
    width: 100%;
    height: 50vh;
    background-position: left 0px top -100px;
    background-size: cover;
`;

const Wrapper = styled(Box)`
    padding: 20px;
    & > h3, & > h5 {
        margin-top: 50px;
  width: '100%',
    height: '50vh',
    objectFit: 'cover'

    }
`;

const Text = styled(Typography)`
    color: #878787;
    margin-top: 10px;
    display: flex;
    flex-direction: row;
    margin-top: 10px;
    display: flex;
    flex-direction: row;

`;
```

```
const Contact = () => {
    return (
        <Box>
            <Banner />
            <Wrapper>
                <Typography variant="h3">Getting in touch is easy!</Typography>
                <Text variant="h5">
                    We are a MCA student from KIET group of institute .And this is our Group
                      Project



          </Text>
            </Wrapper>
        </Box>
    );
}

export default Contact;
```

Create

CreatePost.jsx

```
import React, { useState, useEffect, useContext } from 'react';

import { styled, Box, TextareaAutosize, Button, InputBase, FormControl } from
'@mui/material';
import { AddCircle as Add } from '@mui/icons-material';
import { useNavigate, useLocation } from 'react-router-dom';

import { API } from '../../service/api';
import { DataContext } from '../../context/DataProvider';

const Container = styled(Box)(({ theme }) => ({
   margin: '50px 100px',
   [theme.breakpoints.down('md')]: {
      margin: 0
   }
}));

const Container = styled(Box)(({ theme }) => ({
   margin: '50px 100px',
   [theme.breakpoints.down('md')]: {
      margin: 0
   }
}));


const Image = styled('img')({
   width: '100%',
   height: '50vh',
   objectFit: 'cover'
width: '100%',
   height: '50vh',
   objectFit: 'cover'

});


const StyledFormControl = styled(FormControl)`
   margin-top: 10px;
   display: flex;
   flex-direction: row;
margin-top: 10px;
   display: flex;
   flex-direction: row;

`;
```

```
const Wrapper = styled(Box)`
    padding: 25px 35px;
    display: flex;
    flex: 1;
    overflow: auto;
    flex-direction: column;
    & > div, & > button, & > p {
        margin-top: 20px;
    }

const InputTextField = styled(InputBase)`
    flex: 1;
    margin: 0 30px;
    font-size: 25px;
`;

const Textarea = styled(TextareaAutosize)`
    width: 100%;
    border: none;
    margin-top: 50px;
    font-size: 18px;
    &:focus-visible {
        outline: none;
   width: '100%',
   height: '50vh',
   objectFit: 'cover'

    }
`;

const initialPost = {
    title: '',
    description: '',
    picture: '',
    username: '',
    categories: '',
    createdDate: new Date()
}

const CreatePost = () => {
    const navigate = useNavigate();
    const location = useLocation();

    const [post, setPost] = useState(initialPost);
    const [file, setFile] = useState('');
    const { account } = useContext(DataContext);
const url = post.picture ? post.picture : 'https://images.unsplash.com/photo-1543128639-
4cb7e6eeef1b?ixid=MnwxMjA3fDB8MHxzZWFyY2h8Mnx8bGFwdG9wJTIwc2V0d
XB8ZW58MHx8MHx8&ixlib=rb-1.2.1&w=1000&q=80';

  useEffect(() => {
    const getImage = async () => {
      if(file) {
        const data = new FormData();
        data.append("name", file.name);
        data.append("file", file);

        const response = await API.uploadFile(data);
        post.picture = response.data;
      }
    }
    getImage();
    post.categories = location.search?.split('=')[1] || 'All';
    post.username = account.username;
  }, [file])

  const savePost = async () => {
    await  API.createPost(post);
    navigate('/');
  }

  const handleChange = (e) => {
    setPost({ ...post, [e.target.name]: e.target.value });
  }

  return (
    <Container>
          <Image src={url} alt="post" />

      <StyledFormControl>
        <label htmlFor="fileInput">
          <Add fontSize="large" color="action" />
        </label>
        <input
          type="file"
          id="fileInput"
          style={{ display: "none" }}
          onChange={(e) => setFile(e.target.files[0])}
        />
        <InputTextField onChange={(e) => handleChange(e)} name='title'
placeholder="Title" />
        <Button onClick={() => savePost()} variant="contained"
color="primary">Publish</Button>
      </StyledFormControl>
```

```
            <Textarea
                rowsMin={5}
                placeholder="Tell your story..."
                name='description'
                onChange={(e) => handleChange(e)}
            />
            if(file) {
                const data = new FormData();
                data.append("name", file.name);
                data.append("file", file);

                const response = await API.uploadFile(data);
                if (response.isSuccess) {
                    post.picture = response.data;
                    setImageURL(response.data);
                }
            }

        </Container>
    )
}

export default CreatePost;
```

Update.jsx

```
import React, { useState, useEffect } from 'react';

import { Box, styled, TextareaAutosize, Button, FormControl, InputBase } from
'@mui/material';
import { AddCircle as Add } from '@mui/icons-material';
import { useNavigate, useParams } from 'react-router-dom';

import { API } from '../../service/api';

const Container = styled(Box)(({ theme }) => ({
    margin: '50px 100px',
    [theme.breakpoints.down('md')]: {
        margin: 0
    }
}));

const Image = styled('img')({
    width: '100%',
    height: '50vh',
    objectFit: 'cover'
});

const StyledFormControl = styled(FormControl)`
    margin-top: 10px;
    display: flex;
    flex-direction: row;
`;

const InputTextField = styled(InputBase)`
    flex: 1;
    margin: 0 30px;
    font-size: 25px;
`;

const StyledTextArea = styled(TextareaAutosize)`
    width: 100%;
    border: none;
    margin-top: 50px;
    font-size: 18px;
    &:focus-visible {
        outline: none;
    }
`;

const initialPost = {
    title: '',
    description: '',
    picture: '',
    username: 'codeforinterview',
    categories: 'Tech',
    createdDate: new Date()
}

const Update = () => {
    const navigate = useNavigate();

    const [post, setPost] = useState(initialPost);
    const [file, setFile] = useState('');
    const [imageURL, setImageURL] = useState('');

    const { id } = useParams();

    const url = 'https://images.unsplash.com/photo-1543128639-
4cb7e6eeef1b?ixid=MnwxMjA3fDB8MHxzZWFyY2h8Mnx8bGFwdG9wJTIwc2V0d
XB8ZW58MHx8MHx8&ixlib=rb-1.2.1&w=1000&q=80';

    useEffect(() => {
        const fetchData = async () => {
            let response = await API.getPostById(id);
            if (response.isSuccess) {
                setPost(response.data);
            }
        }
        fetchData();
    }, []);

    useEffect(() => {
        const getImage = async () => {
            if(file) {
                const data = new FormData();
                data.append("name", file.name);
                data.append("file", file);

                const response = await API.uploadFile(data);
                if (response.isSuccess) {
                    post.picture = response.data;
                    setImageURL(response.data);
                }
            }
        }
        getImage();
    }, [file])

    const updateBlogPost = async () => {
        await API.updatePost(post);
        navigate(`/details/${id}`);
    }

    const handleChange = (e) => {
        setPost({ ...post, [e.target.name]: e.target.value });
    }

    return (
        <Container>
if (error.request) {
        // The request was made but no response was received
        console.log("ERROR IN RESPONSE: ", error.toJSON());
        return {
            isError: true,
            msg: API_NOTIFICATION_MESSAGES.requestFailure,
            code: ""
        }
    } else {
        // Something happened in setting up the request that triggered an Error
        console.log("ERROR IN RESPONSE: ", error.toJSON());
        return {
            isError: true,
            msg: API_NOTIFICATION_MESSAGES.networkError,
            code: ""
        }

            <Image src={post.picture || url} alt="post" />

            <StyledFormControl>
                <label htmlFor="fileInput">
                    <Add fontSize="large" color="action" />
                </label>
                <input
                                    type="file"
                    id="fileInput"
                    style={{ display: "none" }}
                    onChange={(e) => setFile(e.target.files[0])}
                />
                <InputTextField onChange={(e) => handleChange(e)} value={post.title}
name='title' placeholder="Title" />
                <Button onClick={() => updateBlogPost()} variant="contained"
color="primary">Update</Button>
            </StyledFormControl>

            <StyledTextArea
                rowsMin={5}
                placeholder="Tell your story..."
                name='description'
                onChange={(e) => handleChange(e)}
                value={post.description}
```

```
          />
          if(file) {
             const data = new FormData();
             data.append("name", file.name);
             data.append("file", file);

             const response = await API.uploadFile(data);
             if (response.isSuccess) {
                post.picture = response.data;
                setImageURL(response.data);
             }
          }
if (error.request) {
       // The request was made but no response was received
       console.log("ERROR IN RESPONSE: ", error.toJSON());
       return {
          isError: true,
          msg: API_NOTIFICATION_MESSAGES.requestFailure,
          code: ""
       }
   } else {
       // Something happened in setting up the request that triggered an Error
       console.log("ERROR IN RESPONSE: ", error.toJSON());
       return {
          isError: true,
          msg: API_NOTIFICATION_MESSAGES.networkError,
          code: ""
       }


       </Container>
   )
}

export default Update;
```

App.js

```
  import { useState } from 'react';


import { Box } from '@mui/material';
import { BrowserRouter, Routes, Route, Navigate, Outlet } from 'react-
router-dom';

//components
import DataProvider from './context/DataProvider';
import Header from './components/header/Header';
import Home from './components/home/Home';
import CreatePost from './components/create/CreatePost';
import DetailView from './components/details/DetailView';
import Update from './components/create/Update';
import About from './components/about/About';
import Contact from './components/contact/Contact';
import Login from './components/account/Login';

const PrivateRoute = ({ isAuthenticated, ...props }) => {
  const token = sessionStorage.getItem('accessToken');
  return isAuthenticated && token ?
    <>
      <Header />
      <Outlet />
    </> : <Navigate replace to='/account' />
};

useEffect(() => {
      const getImage = async () => {
          if(file) {
              const data = new FormData();
              data.append("name", file.name);
              data.append("file", file);

              const response = await API.uploadFile(data);
              post.picture = response.data;
          }
      }
      getImage();
      post.categories = location.search?.split('=')[1] || 'All';
      post.username = account.username;
  }, [file])

  const savePost = async () => {
      await  API.createPost(post);
      navigate('/');
  }
  const handleChange = (e) => {
    setPost({ ...post, [e.target.name]: e.target.value });
  }


function App() {

  const [isAuthenticated, isUserAuthenticated] = useState(false);

  return (
    <DataProvider>
      <BrowserRouter>
        <Box style={{ marginTop: 64 }}>
          <Routes>
            <Route path='/account' element={<Login 
isUserAuthenticated={isUserAuthenticated} />} />

            <Route path='/' element={<PrivateRoute 
isAuthenticated={isAuthenticated} />} >
              <Route path='/' element={<Home />} />
            </Route>

            <Route path='/create' element={<PrivateRoute 
isAuthenticated={isAuthenticated} />} >
              <Route path='/create' element={<CreatePost />} />
            </Route>

            <Route path='/details/:id' element={<PrivateRoute 
isAuthenticated={isAuthenticated} />} >
              <Route path='/details/:id' element={<DetailView />} />
            </Route>

            <Route path='/update/:id' element={<PrivateRoute 
isAuthenticated={isAuthenticated} />} >
              <Route path='/update/:id' element={<Update />} />
            </Route>

            <Route path='/about' element={<PrivateRoute 
isAuthenticated={isAuthenticated} />} >
              <Route path='/about' element={<About />} />
            </Route>

            <Route path='/contact' element={<PrivateRoute 
isAuthenticated={isAuthenticated} />} >
              <Route path='/contact' element={<Contact />} />
            </Route>
          </Routes>
        </Box>
      </BrowserRouter>
    </DataProvider>
  );
}

export default App;
```

App.css

```
  .btn {
  background: unset;
  border: none;
  font-size: 16px;
  text-transform: uppercase;
  font-family: 'Roboto';
  cursor: pointer;
  opacity: 0.8;
}
```

reportWebVitals.js

```
    const reportWebVitals = onPerfEntry => {

  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB
}) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

export default reportWebVitals;
```

Server

controller

comment-controller.js

```
import Comment from '../model/comment.js';


export const newComment = async (request, response) => {
    try {
        const comment = await new Comment(request.body);
        comment.save();

        response.status(200).json('Comment saved successfully');
    } catch (error) {
        response.status(500).json(error);
    }
}


export const getComments = async (request, response) => {
    try {
        const comments = await Comment.find({ postId: request.params.id });

        response.status(200).json(comments);
    } catch (error) {
        response.status(500).json(error)
    }
}

export const deleteComment = async (request, response) => {
    try {
        const comment = await Comment.findById(request.params.id);
        await comment.delete()

        response.status(200).json('comment deleted successfully');
    } catch (error) {
        response.status(500).json(error)
    }
}
```

image-controller.js

```
import grid from 'gridfs-stream';
import mongoose from 'mongoose';

const url = 'http://localhost:8000';


let gfs, gridfsBucket;
const conn = mongoose.connection;
conn.once('open', () => {
   gridfsBucket = new mongoose.mongo.GridFSBucket(conn.db, {
      bucketName: 'fs'
   });
   gfs = grid(conn.db, mongoose.mongo);
   gfs.collection('fs');
});


export const uploadImage = (request, response) => {
   if(!request.file)
      return response.status(404).json("File not found");

   const imageUrl = `${url}/file/${request.file.filename}`;

   response.status(200).json(imageUrl);
}

export const getImage = async (request, response) => {
   try {
      const file = await gfs.files.findOne({ filename: request.params.filename });
      // const readStream = gfs.createReadStream(file.filename);
      // readStream.pipe(response);
      const readStream = gridfsBucket.openDownloadStream(file._id);
      readStream.pipe(response);
   } catch (error) {
      response.status(500).json({ msg: error.message });
   }
}
```

Jwt-controller.js

```
import jwt from 'jsonwebtoken';
import dotenv from 'dotenv';

import Token from '../model/token.js';

dotenv.config();

export const authenticateToken = (request, response, next) => {
    const authHeader = request.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1];

    if (token == null) {
        return response.status(401).json({ msg: 'token is missing' });
    }

    jwt.verify(token, process.env.ACCESS_SECRET_KEY, (error, user) => {
        if (error) {
            return response.status(403).json({ msg: 'invalid token' })
        }

        request.user = user;
        next();
    })
}

export const createNewToken = async (request, response) => {
    const refreshToken = request.body.token.split(' ')[1];

    if (!refreshToken) {
        return response.status(401).json({ msg: 'Refresh token is missing' })
    }

    const token = await Token.findOne({ token: refreshToken });

    jwt.verify(token.token, process.env.REFRESH_SECRET_KEY, (error, user) => {
        if (error) {
            response.status(500).json({ msg: 'invalid refresh token'});
        }
        const accessToken = jwt.sign(user, process.env.ACCESS_SECRET_KEY, {
expiresIn: '15m'});

        return response.status(200).json({ accessToken: accessToken })
    })
```

Post-controller.js

```
import Post from '../model/post.js';


export const createPost = async (request, response) => {
    try {
        const post = await new Post(request.body);
        post.save();

        response.status(200).json('Post saved successfully');
    } catch (error) {
        response.status(500).json(error);
    }
}

export const updatePost = async (request, response) => {
    try {
        const post = await Post.findById(request.params.id);

        if (!post) {
            response.status(404).json({ msg: 'Post not found' })
        }

        await Post.findByIdAndUpdate( request.params.id, { $set: request.body })

        response.status(200).json('post updated successfully');
    } catch (error) {
        response.status(500).json(error);
    }
try {
        const post = await Post.findById(request.params.id);

        if (!post) {
            response.status(404).json({ msg: 'Post not found' })
        }

        await Post.findByIdAndUpdate( request.params.id, { $set: request.body })

        response.status(200).json('post updated successfully');
    } catch (error) {
        response.status(500).json(error);
    }

try {
        const post = await Post.findById(request.params.id);

        if (!post) {
            response.status(404).json({ msg: 'Post not found' })
        }

        await Post.findByIdAndUpdate( request.params.id, { $set: request.body })

        response.status(200).json('post updated successfully');
    } catch (error) {
        response.status(500).json(error);
    }
try {

        const post = await Post.findById(request.params.id);

        if (!post) {
            response.status(404).json({ msg: 'Post not found' })
        }

        await Post.findByIdAndUpdate( request.params.id, { $set: request.body })

        response.status(200).json('post updated successfully');
    } catch (error) {
        response.status(500).json(error);
    }

}

export const deletePost = async (request, response) => {
    try {
        const post = await Post.findById(request.params.id);

        await post.delete()

        response.status(200).json('post deleted successfully');
    } catch (error) {
        response.status(500).json(error)
    }
}

export const getPost = async (request, response) => {
    try {
        const post = await Post.findById(request.params.id);

        response.status(200).json(post);
    } catch (error) {
        response.status(500).json(error)
    }
}

export const getAllPosts = async (request, response) => {
    let username = request.query.username;
    let category = request.query.category;
    let posts;
    try {
        if(username)
            posts = await Post.find({ username: username });
        else if (category)
            posts = await Post.find({ categories: category });
        else
            posts = await Post.find({});

        response.status(200).json(posts);
    } catch (error) {
        response.status(500).json(error)
    }

try {
        if(username)
            posts = await Post.find({ username: username });
        else if (category)
            posts = await Post.find({ categories: category });
        else
            posts = await Post.find({});

        response.status(200).json(posts);
    } catch (error) {
        response.status(500).json(error)
    }

}
```

User-controller.js

```
import bcrypt from 'bcrypt';
import jwt from 'jsonwebtoken';
import dotenv from 'dotenv';

import Token from '../model/token.js'
import User from '../model/user.js';

dotenv.config();
useEffect(() => {
      const getImage = async () => {
         if(file) {
            const data = new FormData();
            data.append("name", file.name);
            data.append("file", file);

            const response = await API.uploadFile(data);
            post.picture = response.data;
         }
      }
      getImage();
      post.categories = location.search?.split('=')[1] || 'All';
      post.username = account.username;
   }, [file])

   const savePost = async () => {
      await  API.createPost(post);
      navigate('/');
   }

   const handleChange = (e) => {
      setPost({ ...post, [e.target.name]: e.target.value });
   }


export const singupUser = async (request, response) => {
   try {
      // const salt = await bcrypt.genSalt();
      // const hashedPassword = await bcrypt.hash(request.body.password, salt);
      const hashedPassword = await bcrypt.hash(request.body.password, 10);

      const user = { username: request.body.username, name: request.body.name,
password: hashedPassword }

      const newUser = new User(user);
      await newUser.save();

      return response.status(200).json({ msg: 'Signup successfull' });
```

```
    } catch (error) {
        return response.status(500).json({ msg: 'Error while signing up user' });
    }
try {
        // const salt = await bcrypt.genSalt();
        // const hashedPassword = await bcrypt.hash(request.body.password, salt);
        const hashedPassword = await bcrypt.hash(request.body.password, 10);

        const user = { username: request.body.username, name: request.body.name,
password: hashedPassword }

        const newUser = new User(user);
        await newUser.save();

        return response.status(200).json({ msg: 'Signup successfull' });
    } catch (error) {
        return response.status(500).json({ msg: 'Error while signing up user' });
    }

}


export const loginUser = async (request, response) => {
    let user = await User.findOne({ username: request.body.username });
    if (!user) {
        return response.status(400).json({ msg: 'Username does not match' });
    }

    try {
        let match = await bcrypt.compare(request.body.password, user.password);

if (match) {

const accessToken = jwt.sign(user.toJSON(), process.env.ACCESS_SECRET_KEY, {
expiresIn: '15m'});

  const refreshToken = jwt.sign(user.toJSON(),
process.env.REFRESH_SECRET_KEY);


 const newToken = new Token({ token: refreshToken });
            await newToken.save();

            response.status(200).json({ accessToken: accessToken, refreshToken:
refreshToken,name: user.name, username: user.username });

        }
 else {
           response.status(400).json({ msg: 'Password does not match' })
        }
    } catch (error) {
        response.status(500).json({ msg: 'error while login the user' })
    }
}

export const logoutUser = async (request, response) => {
    const token = request.body.token;
    await Token.deleteOne({ token: token });

    response.status(204).json({ msg: 'logout successfull' });
}
```

Database

Db.js

```
import mongoose from 'mongoose';

const Connection = async (username, password) => {
   const URL = `mongodb://${username}:${password}@ac-uih4gis-shard-00-
00.dkvcts8.mongodb.net:27017,ac-uih4gis-shard-00-01.dkvcts8.mongodb.net:27017,ac-
uih4gis-shard-00-02.dkvcts8.mongodb.net:27017/?ssl=true&replicaSet=atlas-9yygsb-
shard-0&authSource=admin&retryWrites=true&w=majority` ;
   try {
      await mongoose.connect(URL, { useNewUrlParser: true })
      console.log('Database connected successfully');
   } catch (error) {
      console.log('Error while connecting to the database ', error);
   }
};

export default Connection;
```

Models

Category.js

```
import mongoose from 'mongoose';

const CategorySchema = mongoose.Schema({
    name: {
        type: String,
        required: true
    }
    try {
        let match = await bcrypt.compare(request.body.password, user.password);
        if (match) {
            const accessToken = jwt.sign(user.toJSON(),
process.env.ACCESS_SECRET_KEY, { expiresIn: '15m'});
            const refreshToken = jwt.sign(user.toJSON(),
process.env.REFRESH_SECRET_KEY);

            const newToken = new Token({ token: refreshToken });
            await newToken.save();

            response.status(200).json({ accessToken: accessToken, refreshToken:
refreshToken,name: user.name, username: user.username });

        } else {
            response.status(400).json({ msg: 'Password does not match' })
        }
    } catch (error) {
        response.status(500).json({ msg: 'error while login the user' })
    }
}

export const logoutUser = async (request, response) => {
    const token = request.body.token;
    await Token.deleteOne({ token: token });

    response.status(204).json({ msg: 'logout successfull' });

try {
        // const salt = await bcrypt.genSalt();
        // const hashedPassword = await bcrypt.hash(request.body.password, salt);
        const hashedPassword = await bcrypt.hash(request.body.password, 10);

        const user = { username: request.body.username, name: request.body.name,
password: hashedPassword }

        const newUser = new User(user);
        await newUser.save();
        return response.status(200).json({ msg: 'Signup successfull' });
    } catch (error) {
        return response.status(500).json({ msg: 'Error while signing up user' });
    }
try {
        // const salt = await bcrypt.genSalt();
        // const hashedPassword = await bcrypt.hash(request.body.password, salt);
        const hashedPassword = await bcrypt.hash(request.body.password, 10);

        const user = { username: request.body.username, name: request.body.name,
password: hashedPassword }

        const newUser = new User(user);
        await newUser.save();

        return response.status(200).json({ msg: 'Signup successfull' });
    } catch (error) {
        return response.status(500).json({ msg: 'Error while signing up user' });
    }

});


const category = mongoose.model('category', CategorySchema);

export default category;
```

comment.js

```
import mongoose from 'mongoose';

const CommentSchema = mongoose.Schema({
   name: {
      type: String,
      required: true,
   },
   postId: {
      type: String,
      required: true
   },
   date: {
      type: String,
      required: true
   },
   comments: {
      type: String,
      required: true
   }
});


const comment = mongoose.model('comment', CommentSchema);

export default comment;
```

post.js

```
import mongoose from 'mongoose';

const PostSchema = mongoose.Schema({
   title: {
      type: String,
      required: true,
      unique: true
   },
   description: {
      type: String,
      required: true
   },
   picture:  {
      type: String,
      required: false
   },
   username: {
      type: String,
      required: true
   },
   categories: {
      type: Array,
      required: false
   },
   createdDate: {
      type: Date
   }
});


const post = mongoose.model('post', PostSchema);

export default post;
```

Token.js

```
import mongoose from 'mongoose';

const TokenSchema = mongoose.Schema({
    token: {
        type: String,
        required: true
    }
});


const token = mongoose.model('token', TokenSchema);

export default token;
```

user.js

```
import mongoose from 'mongoose';
const userSchema = mongoose.Schema({
    name: {
        type: String,
        required: true
    },
    username: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true
    }
});
const user = mongoose.model('user', userSchema);
export default user;
```

Routes

Route.js

```
import express from 'express';
import { createPost, updatePost, deletePost, getPost, getAllPosts } from
'../controller/post-controller.js';
import { uploadImage, getImage } from '../controller/image-controller.js';
import { newComment, getComments, deleteComment } from '../controller/comment-
controller.js';
import { loginUser, singupUser, logoutUser } from '../controller/user-controller.js';
import { authenticateToken, createNewToken } from '../controller/jwt-controller.js';
import upload from '../utils/upload.js';
const router = express.Router();
router.post('/login', loginUser);
router.post('/signup', singupUser);
router.post('/logout', logoutUser);
router.post('/token', createNewToken);
router.post('/create', authenticateToken, createPost);
router.put('/update/:id', authenticateToken, updatePost);
router.delete('/delete/:id', authenticateToken, deletePost);

router.get('/post/:id', authenticateToken, getPost);
router.get('/posts', authenticateToken, getAllPosts);

router.post('/file/upload', upload.single('file'), uploadImage);
router.get('/file/:filename', getImage);
router.post('/comment/new', authenticateToken, newComment);
router.get('/comments/:id', authenticateToken, getComments);
router.delete('/comment/delete/:id', authenticateToken, deleteComment);
export default router;
```

utils

upload.js

```
import multer from 'multer';
import { GridFsStorage } from 'multer-gridfs-storage';

const storage = new GridFsStorage({
    url: `mongodb://user:codeforinterview@blogweb-shard-00-
00.ch1hk.mongodb.net:27017,blogweb-shard-00-
01.ch1hk.mongodb.net:27017,blogweb-shard-00-
02.ch1hk.mongodb.net:27017/BLOG?ssl=true&replicaSet=atlas-lhtsci-shard-
0&authSource=admin&retryWrites=true&w=majority`,
    options: { useNewUrlParser: true },
    file: (request, file) => {
        const match = ["image/png", "image/jpg"];

        if(match.indexOf(file.memeType) === -1)
            return`${Date.now()}-blog-${file.originalname}`;

        return {
            bucketName: "photos",
            filename: `${Date.now()}-blog-${file.originalname}`
        }
    }
});

export default multer({storage});
```

Index.js

```
import express from 'express';
import dotenv from 'dotenv';
import cors from 'cors';
import bodyParser from 'body-parser';

//components
import Connection from './database/db.js';
import Router from './routes/route.js';
dotenv.config();
const app = express();

app.use(cors());
app.use(bodyParser.json({ extended: true }));
app.use(bodyParser.urlencoded({ extended: true }));
app.use('/', Router);
const PORT = 8000;
const username = process.env.DB_USERNAME;
const password = process.env.DB_PASSWORD;
Connection(username, password);
app.listen(PORT, () => console.log(`Server is running successfully on PORT
${PORT}`));
```

Dependency file > Package.json

```json
{
  "name": "client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.9.0",
    "@emotion/styled": "^11.8.1",
    "@mui/icons-material": "^5.8.0",
    "@mui/material": "^5.8.0",
    "@testing-library/jest-dom": "^5.14.1",
    "@testing-library/react": "^11.2.7",
    "@testing-library/user-event": "^12.8.3",
    "axios": "^0.21.1",
    "crypto-js": "^4.1.1",
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-router-dom": "^6.3.0",
    "react-scripts": "^5.0.1",
    "web-vitals": "^1.1.2"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
```

Chapter 6

Software Testing

6.1 Software Testing

Software testing is a critical element of software quality assurance and represents the ultimate reuse of specification. Design and code testing represents interesting anomaly for the software during earlier definition and development phase, it was attempted to build software from an abstract concept to tangible implementation. The testing phase involves, testing of the development of the system using various techniques such as White Box Testing, Control Structure Testing.

6.1.1 Testing Technique

6.1.1.1 White Box Testing

White Box Testing is a testing technique in which software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security. In white box testing, code is visible to testers, so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.
It is one of two parts of the Box Testing approach to software testing. Its counterpart, Blackbox testing, involves testing from an external or end-user perspective. On the other hand, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.

6.1.1.2 Black Box Testing

White Box Testing is a testing technique in which software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security. In white box testing, code is visible to testers, so it is also called Clear box

testing, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.
It is one of two parts of the Box Testing approach to software testing. Its counterpart, Blackbox testing, involves testing from an external or end-user perspective. On the other hand, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.

Testing Strategies

A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high level against customer requirements.

1) Unit Testing

2) Integration testing

3) System testing.

4) Acceptance testing.

6.2 Test Scenarios

Test scenarios for blogs can help ensure the functionality, usability, and quality of the blog platform or website. Here are some common test scenarios to consider:

1. User Interface (UI) Testing:

- Verify that the blog has a user-friendly and intuitive interface.

- Check that all navigational elements, such as menus, categories, and tags, are functional and correctly displayed.

- Test the responsiveness of the blog across different devices and screen sizes.

2. Content Creation and Management:

- Test the functionality of creating, editing, and deleting blog posts or articles.

- Verify that the formatting options, such as headings, lists, and links, work correctly.

- Check that media elements like images or videos can be uploaded, displayed, and positioned properly within the blog posts.

- Ensure that tags, categories, and metadata can be added and associated with blog posts.

3. Commenting System:

- Test the functionality of the commenting system, including posting, editing, and deleting comments.

- Verify that comments are properly displayed, nested (if applicable), and sorted chronologically.

- Check for features like comment moderation, spam filtering, and user authentication for commenting.

4. Search and Navigation:

- Test the search functionality to ensure it accurately retrieves relevant blog posts based on keywords or tags.

- Verify that navigation elements like search bars, breadcrumbs, and pagination work as intended.

- Check for the presence and functionality of archive pages, category pages, and tag pages.

5. Social Sharing and Integration:

- Test the sharing functionality to various social media platforms to ensure the sharing links or buttons work correctly.

- Verify the integration of social media widgets or feeds, such as Twitter, Facebook, or Instagram, if present.

6. Performance and Load Testing:

x- Test the blog's performance under different traffic loads to ensure it can handle concurrent users without significant slowdowns or errors.

- Check for the optimal page load times, efficient caching mechanisms, and optimized database queries.

7. Security Testing:

- Verify that user authentication mechanisms, password hashing, and access control are implemented properly.

8. Compatibility Testing:

- Test the blog across different web browsers (Chrome, Firefox, Safari, etc.) and ensure consistent functionality and appearance.

- Verify compatibility with different operating systems and devices, including desktops, laptops, tablets, and smartphones.

These test scenarios provide a starting point for testing the functionality and usability of a blog platform or website. The actual test scenarios may vary depending on the specific features and requirements of your blog.

Test Case of Blog Website

Test Case of Login Page:

Project Name	Blog Website
Module Name	Login (Login page)
Created By	Shivam Kumar
Created Date	25-05-2023
Reviewed By	Group Member
Reviewed Date	20-05-2023

Test Scenario ID	TS_BW_001	TS_BW_002	TS_BW_001
Test case ID	TC_BW_Login001	TC_BW_Login002	TC_BW_Login003

Test case description	Enter valid user name and valid password and Login	Enter valid user name and Invalid password and Login	Enter Invalid user name and valid password and Login
Test Stpes	Valid URL Test Data	Valid URL Test Data	Valid URL Test Data
Test Scenario ID	Succesfull Login	Get a POP message to show an error that you have entered an invalid username or password	Get a POP message to show an error that you have entered an invalid username or password
Test Scenario ID	Shivam Kumar	Vikash	Piyush
Test Scenario ID	28-05-2023	28-05-2023	28-05-2023
Test Scenario ID	No Comments	No Comments	No Comments

Test Case of Home Page Functionality

Project Name	Blog Website
Module Name	**Home Page Functionality**
Created By	Shivam Kumar
Created Date	25-05-2023
Reviewed By	Group Members
Reviewed Date	25-05-2023

Test Scenario ID	HOMEPAGE_TC001
Test Case Description:	This test case verifies the basic functionality and layout of the home page.
Preconditions:	1. The website is accessible and functional. 2.The user has a compatible web browser.

Test Steps:	1.Open the web browser. 2.Enter the URL of the website. 3.Press Enter or click the "Go" button. 4.Wait for the home page to load.
Expected Results:	The home page should load without any errors. The home page content should be displayed properly. The layout and design elements should be visually appealing and consistent. The home page should have a clear and easily navigable menu or navigation bar. The home page should contain relevant and informative content. The home page should load within a reasonable time frame.
Test Data:	URL: http://localhost:3000/account
Executed by	Shivam Kumar
Executed date	28-05-2023

Test Case of Category Page Functionality

Project Name	Blog Website
Module Name	Category Page Functionality
Created By	Shivam Kumar
Created Date	25-05-2023
Reviewed By	Group Members
Reviewed Date	25-05-2023

Test Scenario ID	CATEGORY_TC001
Test Case Description:	This test case verifies the functionality and behavior of the category page on a website.

Preconditions:	The website is accessible and functional. The user has a compatible web browser. The category page exists and is accessible from the home page or navigation menu.
Test Steps:	Open the web browser. Enter the URL of the website. Press Enter or click the "Go" button. Wait for the home page to load. Navigate to the category page by clicking on the category link or selecting it from the navigation menu. Wait for the category page to load.
Expected Results:	The home page should load without any errors. The category page should load without any errors. The category page should display the relevant content based on the selected category. The layout and design elements on the category page should be visually appealing and consistent with the overall website design. The category page should contain a clear and easily navigable menu or navigation bar. The category page should display a list of items or subcategories related to the selected category. If subcategories are present, clicking on a subcategory should navigate to the respective subcategory page. The category page should have proper pagination if the number of items or subcategories exceeds the page limit. Clicking on an item within the category should navigate to the respective item page. The category page should load within a reasonable time frame.
Test Data:	URL: http://localhost:3000/?category=Music
Executed by	Piyush Rajput
Executed date	28-05-2023

Test Case of Comment Section Functionality

Project Name	Blog Website
Module Name	Comment Section Functionality
Created By	Shivam Kumar

Created Date	25-05-2023
Reviewed By	Group Members
Reviewed Date	25-05-2023

Test Scenario ID	COMMENT_TC001
Test Case Description:	This test case verifies the functionality and behavior of the comment section on a website or webpage.
Preconditions:	1.The website or webpage containing the comment section is accessible and functional. 2.The user has a compatible web browser. 3.The comment section exists and is visible on the webpage.
Test Steps:	Open the web browser. Enter the URL of the website or webpage containing the comment section. 1.Press Enter or click the "Go" button. 2.Wait for the webpage to load. 3.croll down to locate the comment section. 4.Read any existing comments, if present.
Expected Results:	The webpage should load without any errors. The comment section should be displayed on the webpage. Existing comments, if present, should be visible and properly formatted. The comment section should have a clear and easily identifiable form for submitting new comments. The comment form should include input fields for the user's name, email (optional), and the comment text. The comment form should include a "Submit" button for posting the comment. Upon submitting a comment, the comment should be displayed in the comment section along with the user's name and timestamp. The comment section should support basic formatting options such as line breaks, bold text, italic text, etc., if applicable. The comment section should have a "Reply" or "Reply to Comment" feature, if applicable. The comment section should allow users to edit or delete their own comments, if applicable. The comment section should have pagination or load more functionality if the number of comments exceeds the page limit. The comment section should have proper moderation and spam protection measures in place, if applicable.

	The comment section should display an appropriate error message if there are any issues with submitting or displaying comments. The comment section should load within a reasonable time frame.
Test Data:	URL: www.example.com/page (replace with the actual URL of the webpage containing the comment section being tested) Commenter Name: [Specify a name for testing] Commenter Email: [Specify an email address for testing] Comment Text: [Specify a comment text for testing]
Executed by	Piyush Rajput
Executed date	28-05-2023

Test Case of Reading Comment Section:

Project Name	Blog Website
Module Name	Reading Comment Section
Created By	Shivam Kumar
Created Date	25-05-2023
Reviewed By	Test Lead/Peers
Reviewed Date	29-05-2023

Test Scenario ID	COMMENT_READ_TC001
Test Case Description:	This test case verifies the functionality of reading and viewing comments in the comment section of a website or webpage
Preconditions:	1.The website or webpage containing the comment section is accessible and functional. 2.The user has a compatible web browser. 3.The comment section exists and contains at least one comment.
Test Steps:	Open the web browser. Enter the URL of the website or webpage containing the comment section. 1.Press Enter or click the "Go" button.

	2. Wait for the webpage to load. 3. Scroll down to locate the comment section. 4. Read the existing comments in the comment section.
Expected Results:	The webpage should load without any errors. The comment section should be displayed on the webpage. The existing comments should be visible and properly formatted. Each comment should display the commenter's name, timestamp, and comment text. Comments should be displayed in chronological order, with the most recent comment appearing at the top. Long comments should be properly truncated or have an option to expand for readability. If there are multiple pages of comments, the pagination or load more functionality should be visible and functional. If comments are nested or have replies, the nested structure should be clearly indicated for better readability. If comments have upvotes or downvotes, the vote counts should be displayed next to each comment, if applicable. The comment section should display an appropriate message if no comments are present. The comment section should load within a reasonable time frame.
Test Data:	URL: http://localhost:3000/commentsec
Executed by	Piyush Rajput
Executed date	29-05-2023

Future Enhancement

This project was developed to fulfil the requirement of common blogging platform for educational purposes. However, there are lots of scope to improve the performance of the Online Blogging System in the area of user interface, database performance, and query processing time etc. So, there are many things for future enhancement of this project. The future enhancements that are possible in the project are as follows:

1 Possibility of account creation of parents/guardians with limited rights.

2 Web based interface for generate reports, like who has published more contents, who has commented most, the logged in time etc.

3 Linking and integration of other online educational web sites.

4 Integration with school/college database through Web Services.

5 Development of mobile application which can run on multiple OS and devices.

6 We can also add a quiz feature and winners will get some educational assistance or any other reward.

Bibliography

Websites

Following websites has been referred to create this project report:

1 . https:// react.dev/

2 . https://www.mongodb.com/docs/atlas/getting-started/

3. https://expressjs.com/en/5x/api.html

4 . https://mongoosejs.com/docs/

Books/e-books

Following books/e-books has been referred to create this project report:

1. Major Project Guideline.pdf