

Our website is a user-driven news blog where individuals can directly post their news articles. It's a platform for sharing diverse perspectives and engaging with news content across various topics.

**AN INTERIM PROJECT REPORT**

*Submitted by*

*Abhijeet Kumar- 2051017*

*Harsh Raj- 2051030*

*Tushar Agarwal- 2051025*

*Riya Singh- 2051261*

Under the supervision of

***Prof. Jhalak Dutta***

*Department of Computer Science and Engineering* In partial fulfilment for the award of the degree Of

**BACHELOR OF TECHNOLOGY**

In

**COMPUTER SCIENCE AND ENGINEERING**



### HERITAGE INSTITUTE OF TECHNOLOGY, KOLKATA

An Autonomous Institution affiliated to MAKAUT, West Bengal

ACKNOWLEDGMENT

Apart from the efforts, the success of our project depend slargely on the encouragement and guidelines of many others. We take this opportunity to express our gratitude to all those who have been helpful in the successful completion of this end eavor.

We would like to express our deep sense of thanks and gratitude to our mentor, supervisor and teacher

Prof. Jhalak Dutta, from the Computer Science and Engineering Department, Heritage Institute of Technology, for providing his invaluable guidance, comments and suggestions during the course of this project.

We would also like to thank our teammates for the constant source of motivation and dedication put into the project.

INDEX

1. **Introduction**
   1. [Problem Statement. 4](#_TOC_250008)
   2. [Project Overview 4](#_TOC_250007)
2. [Project Details](#_TOC_250006)
   1. [Project Name 5](#_TOC_250005)
   2. [Tools and Technologies 5](#_TOC_250004)
   3. [Actors. 6](#_TOC_250003)
   4. [Features 6](#_TOC_250002)
   5. [Implementation Details 7](#_TOC_250001)
   6. [UML Diagrams](#_TOC_250000)
      1. UseCase Diagram… 8
      2. Sequence Diagram. 9
      3. Data Flow 12
3. Project Screenshots 15
4. Project Codes 18
5. Further Improvement 33
6. Conclusion 33
7. Bibliography 33

**INTRODUCTION**

## Problem Statement:

In the ever-expanding digital landscape, online audiences are consistently increasing their engagement with news content, spending an average of more than 2 hours daily. Prominent news platforms, such as CNN and BBC, have witnessed substantial growth, with viewership skyrocketing tenfold since their establishment in the early 1980s.

However, a significant challenge faced by news consumers is the lack of user-friendly platforms to find news articles aligned with their specific interests. Existing news blogs often offer disorganized and cluttered user experiences, making the process of discovering noteworthy content cumbersome.

To tackle this challenge, we've introduced a user-centric news blog platform. Unlike traditional news blogs, our platform allows users to directly contribute and share their news articles without relying on external APIs. This approach aims to create a seamless and personalized news-sharing experience, empowering users to easily post, explore, and engage with diverse news content across specific categories.

# Project Overview

**‘News Times’** is a web-based news blog platform designed to streamline the exploration of diverse news content while prioritizing an intuitive user experience. The application empowers users to contribute and share news articles across various categories, including sports, entertainment, politics, education, and more. Each news article is accompanied by detailed information, including headlines, summaries, and source details.

To ensure a seamless user interface, the platform is developed using a combination of **HTML, CSS,** and **JavaScript**. The dynamic and responsive nature of the website is further enhanced through the use of **ReactJS**, providing users with an interactive and engaging experience. The application leverages **MongoDB** as the database to efficiently store and retrieve user-contributed news articles.

The back-end functionality is powered by **Node.js** and **Express.js**, enabling robust server-side operations. Through this technology stack, the News Blog allows users to easily browse, filter, and access news content tailored to their specific interests, fostering a user-driven and collaborative news-sharing community.

# PROJECT DETAILS

In this section, we will discuss in details the project features, the various technologies used in this project, the actors (or users) of this application, implementation details and further modifications.

## Project Name

**‘News Times’**  is an online news blog platform, providing a unified space for diverse users to seamlessly search, explore, and uncover news content tailored to their specific preferences. It simplifies the process of engaging with news, offering a user-friendly interface for individuals to browse and discover articles aligned with their interests.

## Tools and Technologies

* + - Operating System– Windows 11 Home
    - Internet Browser– Google Chrome (or any latest browser)
    - Visual Studio Code–IDE
    - HTML
    - CSS
    - JavaScript
    - React.js
    - Express.js
    - Node.js
    - MongoDB

## Actors

**Users:** These are individuals who visit the news blog website to explore and read news articles across categories such as sports, entertainment, politics, and education. Users are required to sign up or sign in to access features like contributing articles, engaging in discussions, and receiving personalized recommendations.

**News Posting Users:** Individuals who contribute news articles to the platform. They are also required to sign up or sign in to share their articles with the community.

HTML, CSS, and JavaScript Front-End: These technologies collectively form the user interface of the news blog website. HTML structures the content, CSS styles the presentation, and JavaScript handles dynamic behavior, such as loading articles and providing user interaction.

ReactJS, MongoDB, Node.js, Express.js Back-End: These technologies power the back-end of the news blog, handling server-side operations, database storage (MongoDB), and ensuring a dynamic and responsive user experience (ReactJS).

**Features:**

**i. Discover News Content:** Users can effortlessly explore diverse news articles across categories like sports, entertainment, politics, and more after signing up or signing in.

**ii. News Highlights:** The platform showcases a section dedicated to news highlights, offering users top recommendations and trending articles in their selected categories, accessible after login.

**iii. User-Friendly Interface:** The website boasts a simple and intuitive graphical user interface, allowing easy navigation and access to news content. Both news reading/exploring users and news posting users need to sign up or sign in for a personalized experience.

**iv. Custom API Integration:** The platform seamlessly integrates internal databases (MongoDB) to store and retrieve user-contributed news articles, ensuring a personalized and collaborative news-sharing experience. Access to this feature is granted upon login.

**Implementation Details:**

**User Access:** Both news reading/exploring users and news posting users need to sign up or sign in to access features like contributing articles, engaging in discussions, and receiving personalized recommendations.

**Home Page:** Upon logging in, users are directed to the homepage, where they can seamlessly perform various actions, including browsing news articles, selecting specific categories, and accessing trending news highlights.

**Search Functionality:** Users can conduct searches for news content based on keywords or categories. The search queries are processed on the front-end using ReactJS, leveraging the internal database (MongoDB) to fetch relevant news articles.

**News Highlights:** The website features a section providing real-time news highlights and trending articles in various categories. This content is retrieved from the internal database (MongoDB) and displayed on the homepage, accessible after logging in.

The news blog website prioritizes a personalized user experience for both news reading/exploring users and news posting users, requiring them to sign up or sign in to access the full range of features and contribute to the collaborative news-sharing community.

.

## UMLDiagrams

**Use Cases Diagram:**

In the News Blog Website, users can perform various actions, each initiated by the user. The interactions are facilitated through a Node.js server, communicating with an API server to fetch the necessary data from the underlying database. Noteworthy interactions include login and register processes, where authorization and authentication are handled.

**Browse News Articles:**

**Description:** Users can explore a variety of news articles across different categories.

**Initiated By:** User

**Communication:** Node.js server communicates with the API server to fetch relevant news data.

**Search for News:**

**Description:** Users can conduct searches based on keywords or categories to find specific news articles.

**Initiated By:** User

**Communication:** Node.js server interacts with the API server to process and retrieve search results from the database.

**View News Highlights:**

**Description:** Users can access a section showcasing top news highlights and trending articles.

**Initiated By:** User

**Communication:** Node.js server communicates with the API server to fetch real-time highlights from the database.

**User Registration:**

**Description:** New users can register for an account on the news blog website.

**Initiated By:** User

**Communication:** Node.js server manages the registration process, ensuring user details are securely stored in the database.

**User Login:**

**Description:** Registered users can log in to access personalized features and contribute to the platform.

**Initiated By:** User

**Communication:** Node.js server handles the login process, authenticating users through the API server.

**Contribute News Article:**

**Description:** Registered users can submit and contribute their news articles to the platform.

**Initiated By:** User

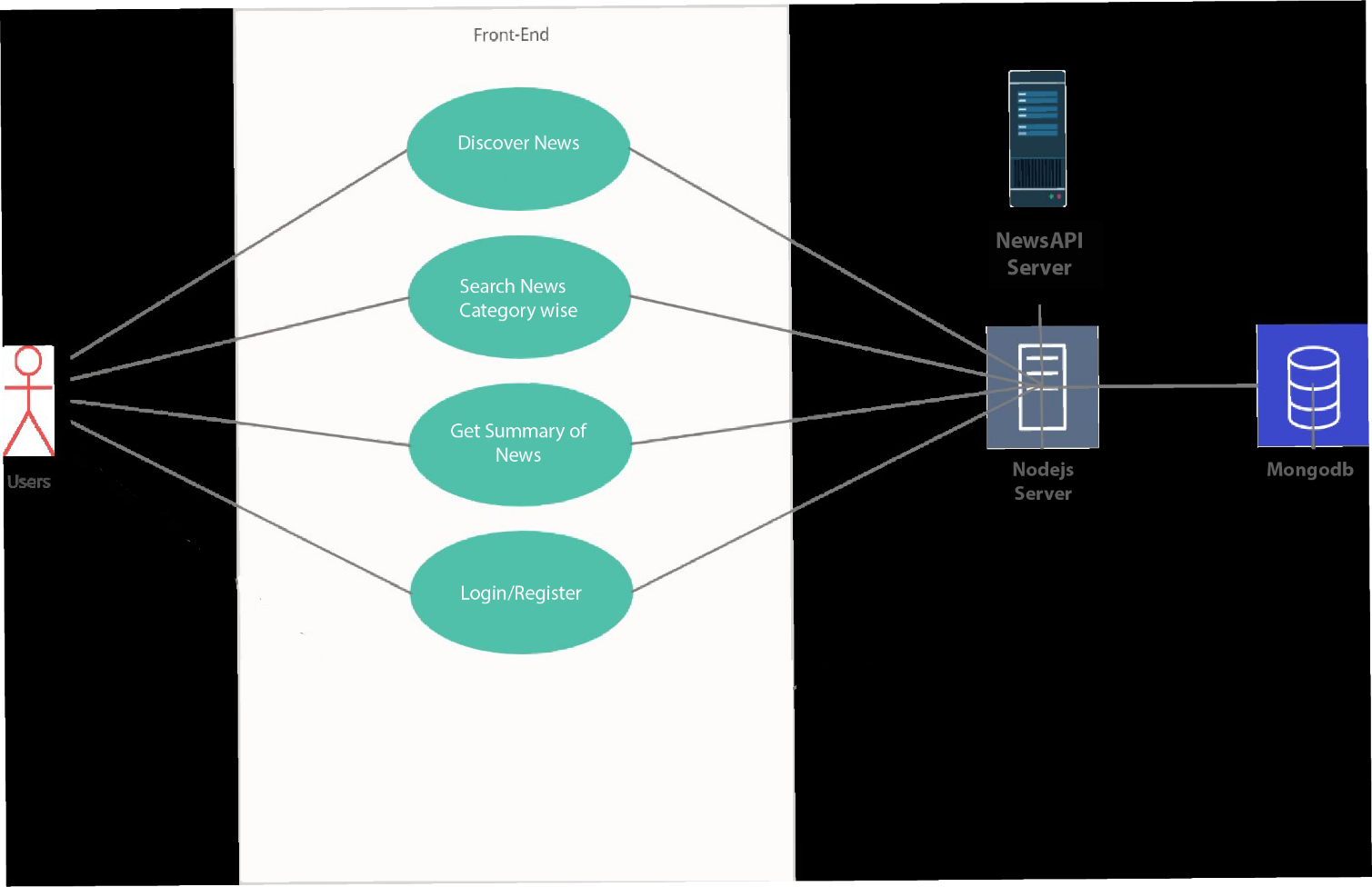
**Communication:** Node.js server communicates with the API server to store the contributed article in the database.

**Engage in Discussions:**

**Description:** Users can participate in discussions related to news articles.

**Initiated By:** User

**Communication:** Node.js server interacts with the API server to enable users to engage in discussions and share opinions.

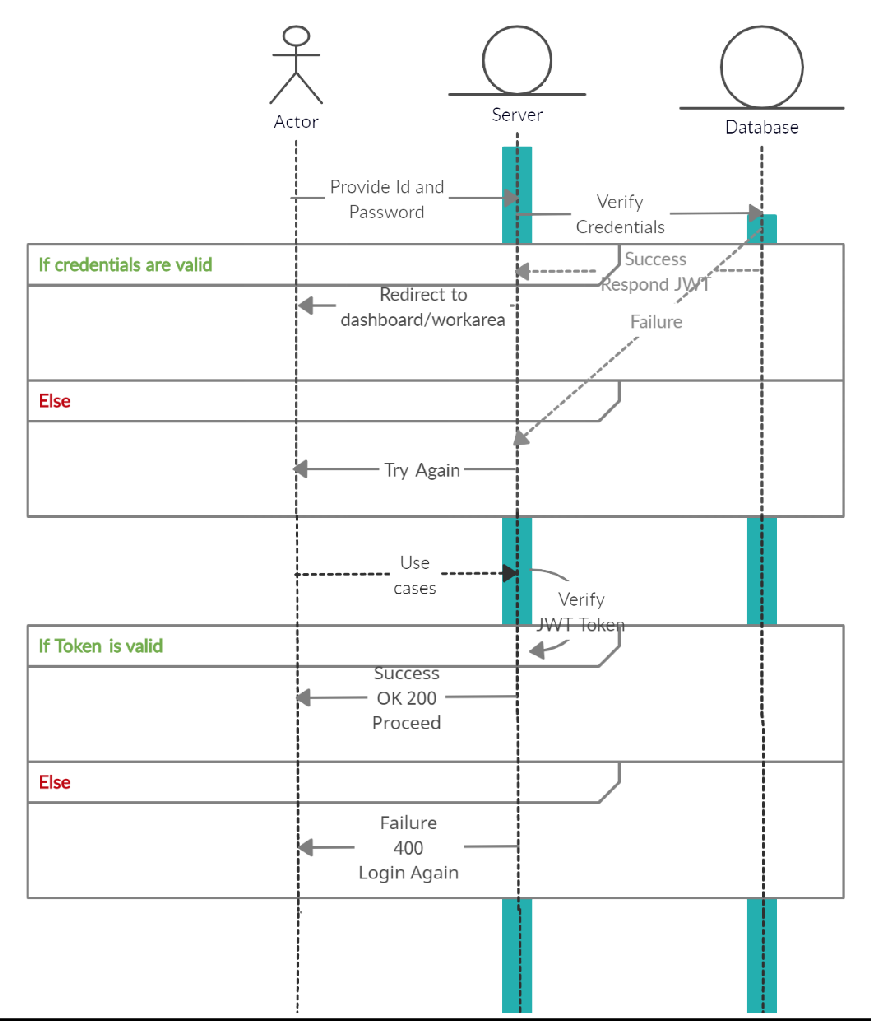
These use cases illustrate the user-initiated actions in the News Blog Website, highlighting the seamless communication between the Node.js server and the API server to facilitate a dynamic and interactive user experience. Authorization and authentication are specifically managed during user registration and login processes to ensure secure access to personalized features.

*Usecase diagram: User actions*

## Sequence Diagram:

Authentication of Users

Actors have to provide an email and a password to access the application. Upon successful login/registration, a JWT is provided as a response to the actor(which is stored in Cookies).

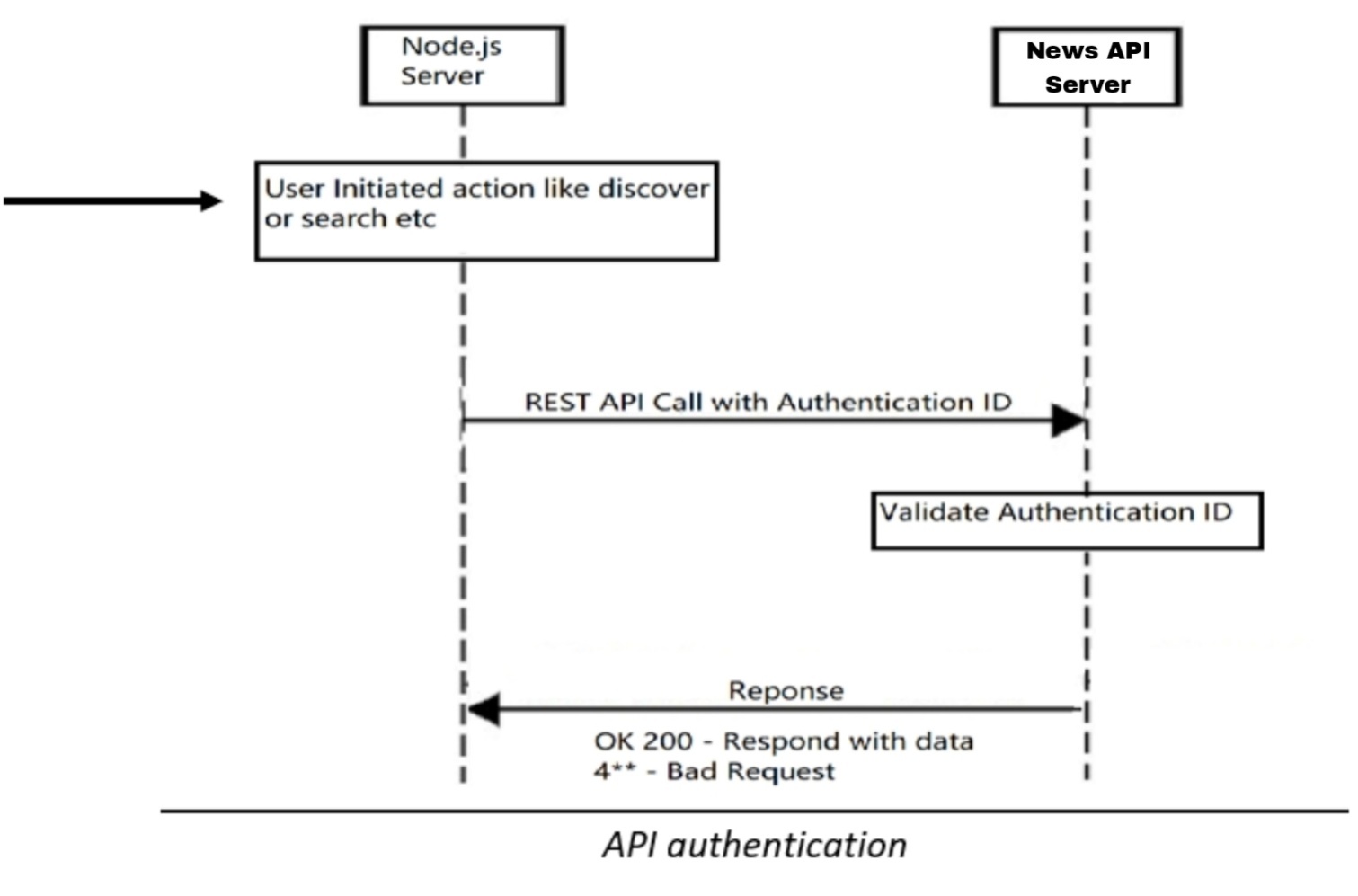


*Authentication of Users*

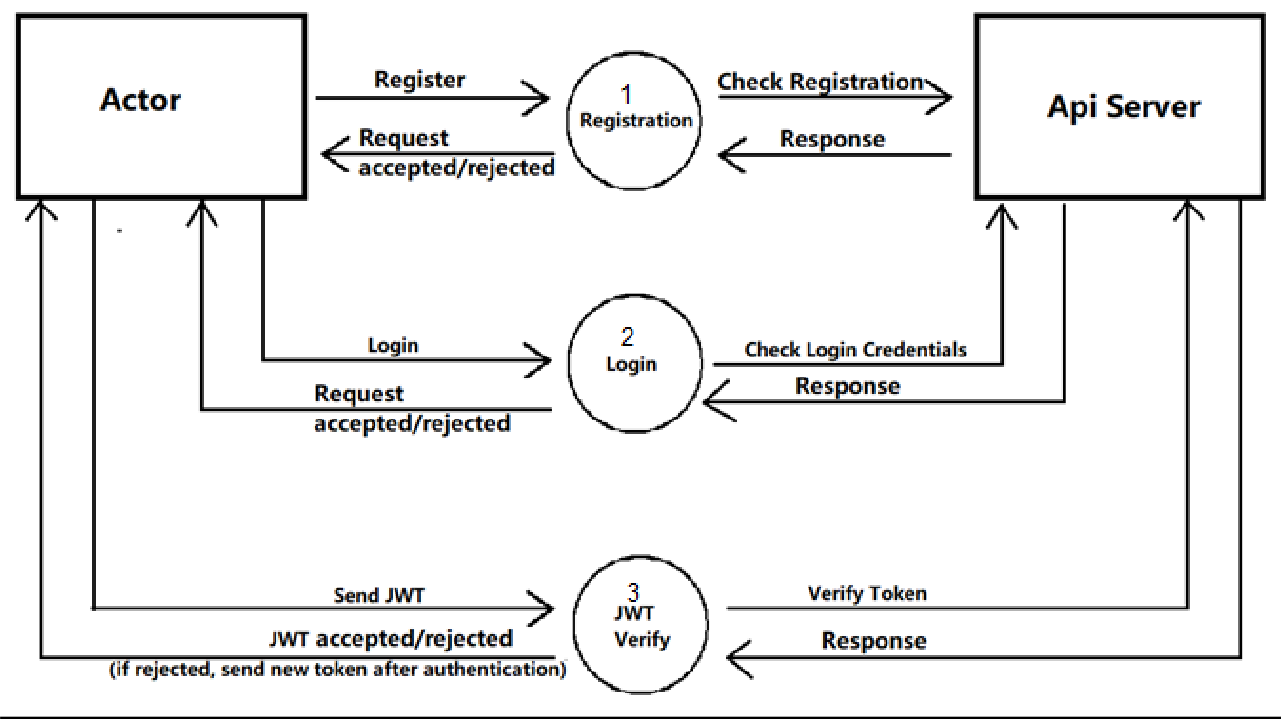
**Data Retrieval from MongoDB:**

As the News Blog Website does not rely on external APIs for fetching data, but instead utilizes a MongoDB database to store and retrieve news content, the focus shifts to ensuring secure and authorized access to the internal database.

*API authentication*



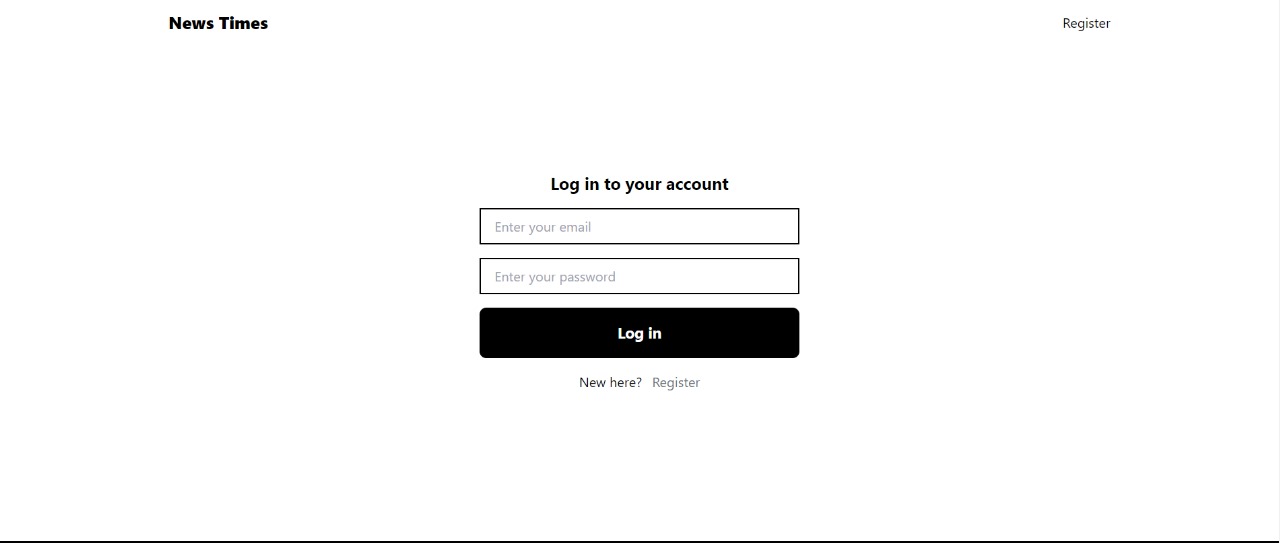
## DataFlow Diagram:



*Authorization and authentication of User*

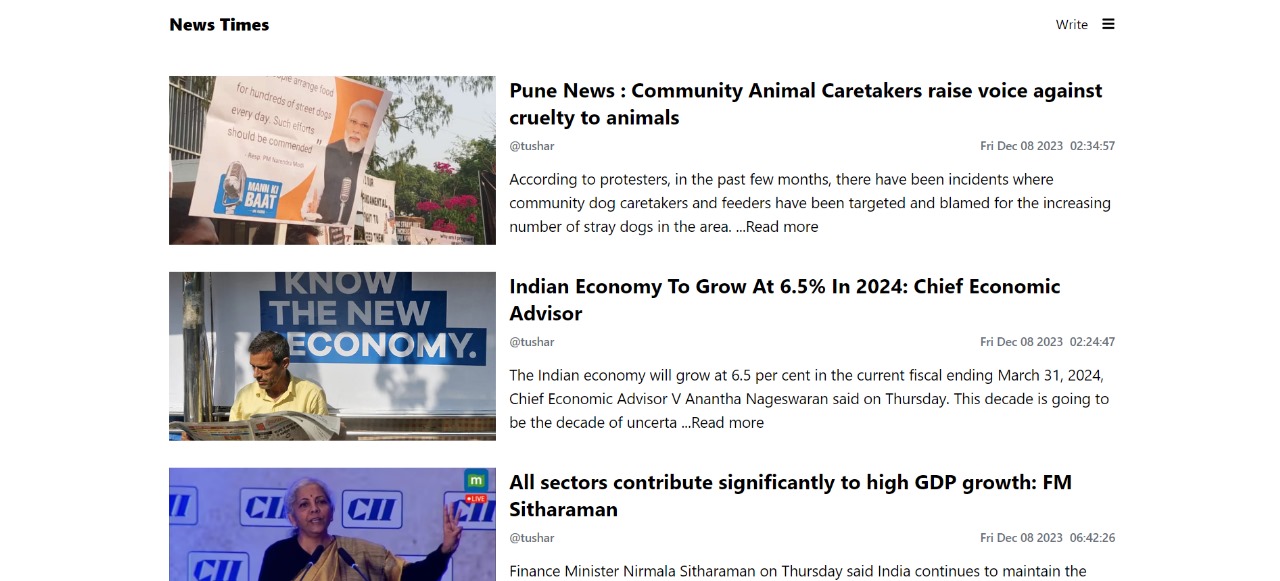
# Project Screenshots

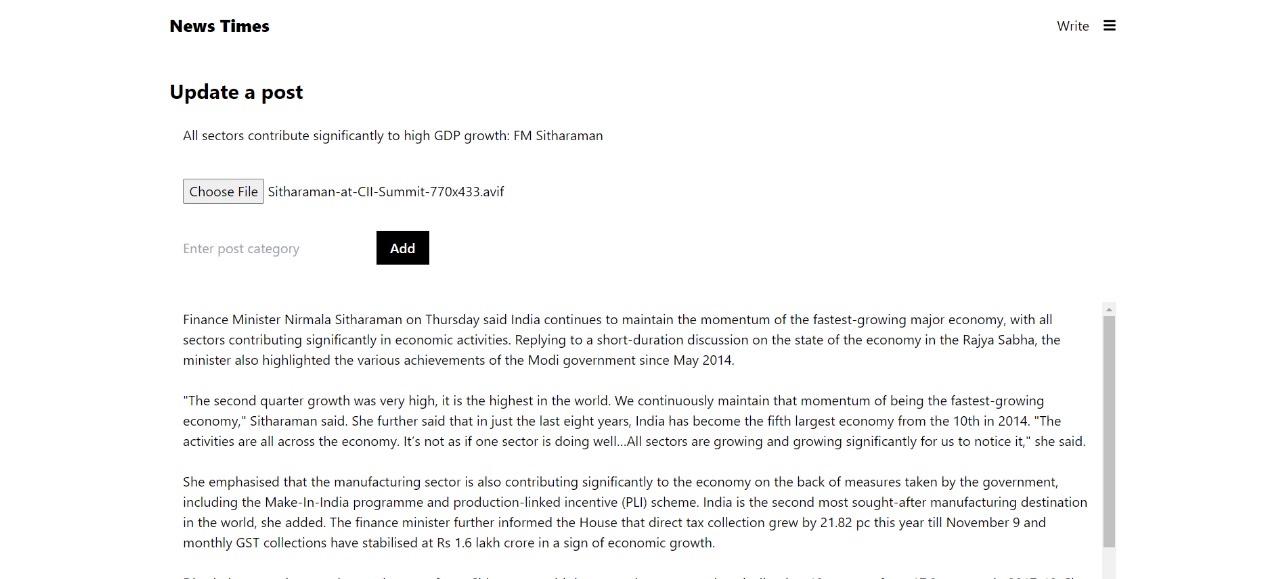
# Login Page

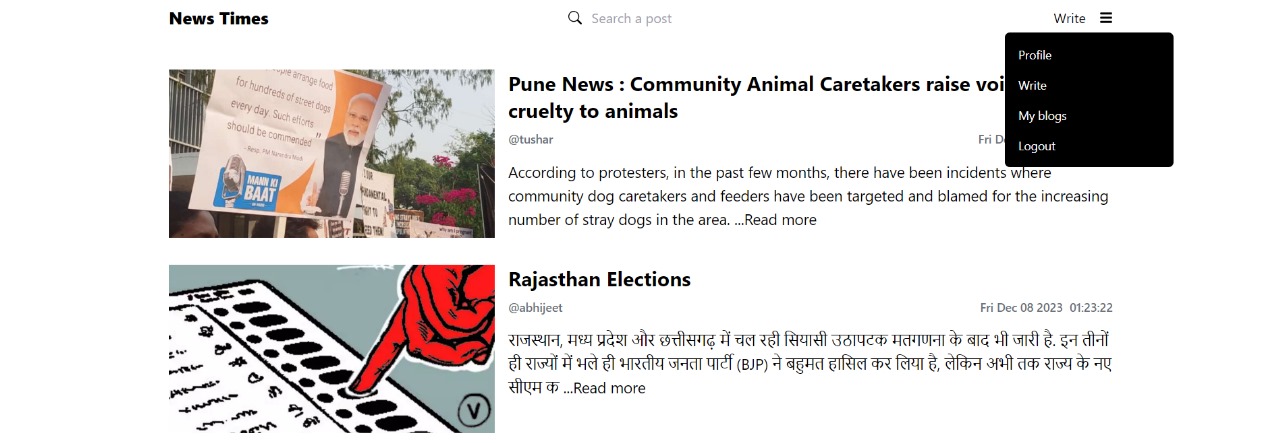
****

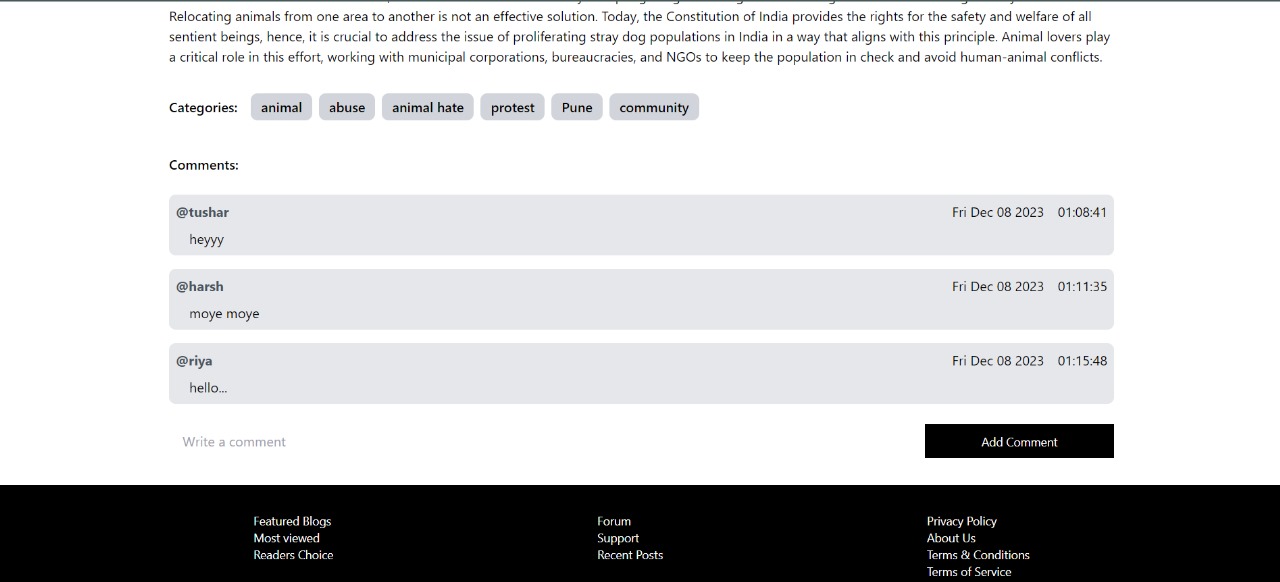
***Post a news article***

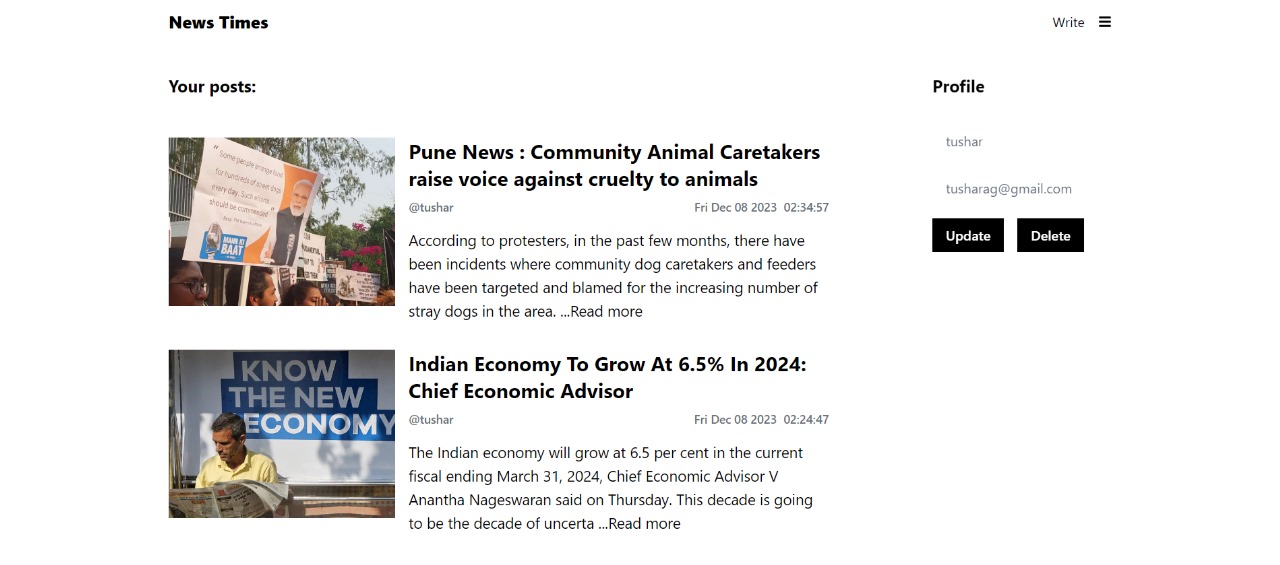
**

***News Articles***

***Update the post***

**Menu Bar**

**Post and delete Comment**

**User Profile Page**

# Project Code

# Frontend Code:

### Index.html: Dynamic HTML code which renders the basic structure of webpage

<!doctype html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <link rel="icon" type="image/svg+xml" href="/vite.svg" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Blog Market</title>

  </head>

  <body>

    <div id="root"></div>

    <script type="module" src="/src/main.jsx"></script>

  </body>

</html>

**App.jsx:** acts as the orchestrator for the News Blog Website, defining routes, managing user context, and rendering components based on user interactions. Its structure and logic contribute to an effective and user-friendly experience on the website.

import {Route, Routes} from 'react-router-dom'

import Home from "./pages/Home"

import Login from "./pages/Login"

import Register from "./pages/Register"

import PostDetails from './pages/PostDetails'

import CreatePost from './pages/CreatePost'

import EditPost from './pages/EditPost'

import Profile from './pages/Profile'

import {  UserContextProvider } from './context/UserContext'

import MyBlogs from './pages/MyBlogs'

const App = () => {

  return (

      <UserContextProvider>

      <Routes>

      <Route exact path="/" element={<Home/>}/>

      <Route exact path="/login" element={<Login/>}/>

      <Route exact path="/register" element={<Register/>}/>

      <Route exact path="/write" element={<CreatePost/>}/>

      <Route exact path="/posts/post/:id" element={<PostDetails/>}/>

      <Route exact path="/edit/:id" element={<EditPost/>}/>

      <Route exact path="/myblogs/:id" element={<MyBlogs/>}/>

      <Route exact path="/profile/:id" element={<Profile/>}/>

      </Routes>

      </UserContextProvider>

  )

}

export default App

### main.jsx:

import ReactDOM from 'react-dom/client'

import App from './App.jsx'

import './index.css'

import { BrowserRouter } from 'react-router-dom'

ReactDOM.createRoot(document.getElementById('root')).render(

  <BrowserRouter>

    <App />

  </BrowserRouter>

)

### Comment.jsx : To post the comments on an news article

import axios from "axios"

import { BiEdit } from "react-icons/bi"

import { MdDelete } from "react-icons/md"

import { URL } from "../url"

import { useContext } from "react"

import { UserContext } from "../context/UserContext"

const Comment = ({c,post}) => {

  const {user}=useContext(UserContext)

  const deleteComment=async(id)=>{

    try{

      await axios.delete(URL+"/api/comments/"+id,{withCredentials:true})

      window.location.reload(true)

    }

    catch(err){

      console.log(err)

    }

  }

  // console.log(post.userId)

  // console.log(user.\_id)

  // console.log(post)

  // console.log(user)

  return (

    <div className="px-2 py-2 bg-gray-200 rounded-lg my-2">

           <div className="flex items-center justify-between">

            <h3 className="font-bold text-gray-600">@{c.author}</h3>

            <div className="flex justify-center items-center space-x-4">

            <p>{new Date(c.updatedAt).toString().slice(0,15)}</p>

            <p>{new Date(c.updatedAt).toString().slice(16,24)}</p>

            {user?.\_id===c?.userId ?

              <div className="flex items-center justify-center space-x-2">

                    <p className="cursor-pointer" onClick={()=>deleteComment(c.\_id)}><MdDelete/></p>

                </div>:""}

            </div>

           </div>

           <p className="px-4 mt-2">{c.comment}</p>

           </div>

  )

}

export default Comment

**ProfilePosts.jsx: for user profile page**

/\* eslint-disable react/prop-types \*/

import {IF} from '../url'

const ProfilePosts = ({p}) => {

  // console.log(p)

  return (

    <div className="w-full flex mt-8 space-x-4">

    {/\* left \*/}

    <div className="w-[35%] h-[200px] flex justify-center items-center">

    <img src={IF+p.photo} alt="" className="h-full w-full object-cover"/>

    </div>

    {/\* right \*/}

    <div className="flex flex-col w-[65%]">

      <h1 className="text-xl font-bold md:mb-2 mb-1 md:text-2xl">

      {p.title}

      </h1>

      <div className="flex mb-2 text-sm font-semibold text-gray-500 items-center justify-between md:mb-4">

       <p>@{p.username}</p>

       <div className="flex space-x-2">

       <p>{new Date(p.updatedAt).toString().slice(0,15)}</p>

       <p>{new Date(p.updatedAt).toString().slice(16,24)}</p>

       </div>

      </div>

      <p className="text-sm md:text-lg">{p.desc.slice(0,200)+" ...Read more"}</p>

    </div>

    </div>

  )

}

export default ProfilePosts

**Backend Code:**

**Index.js:**

const express=require('express')

const app=express()

const mongoose=require('mongoose')

const dotenv=require('dotenv')

const cors=require('cors')

const multer=require('multer')

const path=require("path")

const cookieParser=require('cookie-parser')

const authRoute=require('./routes/auth')

const userRoute=require('./routes/users')

const postRoute=require('./routes/posts')

const commentRoute=require('./routes/comments')

//database

const connectDB=async()=>{

    try{

        await mongoose.connect(process.env.MONGO\_URL)

        console.log("database is connected successfully!")

    }

    catch(err){

        console.log(err)

    }

}

//middlewares

dotenv.config()

app.use(express.json())

app.use("/images",express.static(path.join(\_\_dirname,"/images")))

app.use(cors({origin:"http://localhost:5173",credentials:true}))

app.use(cookieParser())

app.use("/api/auth",authRoute)

app.use("/api/users",userRoute)

app.use("/api/posts",postRoute)

app.use("/api/comments",commentRoute)

//image upload

const storage=multer.diskStorage({

    destination:(req,file,fn)=>{

        fn(null,"images")

    },

    filename:(req,file,fn)=>{

        fn(null,req.body.img)

        // fn(null,"image1.jpg")

    }

})

const upload=multer({storage:storage})

app.post("/api/upload",upload.single("file"),(req,res)=>{

    // console.log(req.body)

    res.status(200).json("Image has been uploaded successfully!")

})

app.listen(process.env.PORT,()=>{

    connectDB()

    console.log("app is running on port "+process.env.PORT)

})

Comment.js:

const mongoose=require('mongoose')

const CommentSchema=new mongoose.Schema({

    comment:{

        type:String,

        required:true,

    },

    author:{

        type:String,

        required:true,

    },

    postId:{

        type:String,

        required:true,

    },

    userId:{

        type:String,

        required:true

    }

},{timestamps:true})

module.exports=mongoose.model("Comment",CommentSchema)

**Post.js:**

const mongoose=require('mongoose')

const PostSchema=new mongoose.Schema({

    title:{

        type:String,

        required:true,

        unique:true

    },

    desc:{

        type:String,

        required:true,

        unique:true

    },

    photo:{

        type:String,

        required:false,

    },

    username:{

        type:String,

        required:true,

    },

    userId:{

        type:String,

        required:true,

    },

    categories:{

        type:Array,

    },

},{timestamps:true})

module.exports=mongoose.model("Post",PostSchema)

**User.js:**

const mongoose=require('mongoose')

const UserSchema=new mongoose.Schema({

    username:{

        type:String,

        required:true,

        unique:true

    },

    email:{

        type:String,

        required:true,

        unique:true

    },

    password:{

        type:String,

        required:true,

    }

},{timestamps:true})

module.exports=mongoose.model("User",UserSchema)

**auth.js:** Code related to authentication page

const express=require('express')

const router=express.Router()

const User=require('../models/User')

const bcrypt=require('bcrypt')

const jwt=require('jsonwebtoken')

//REGISTER

router.post("/register",async(req,res)=>{

    try{

        const {username,email,password}=req.body

        const salt=await bcrypt.genSalt(10)

        const hashedPassword=await bcrypt.hashSync(password,salt)

        const newUser=new User({username,email,password:hashedPassword})

        const savedUser=await newUser.save()

        res.status(200).json(savedUser)

    }

    catch(err){

        res.status(500).json(err)

    }

})

//LOGIN

router.post("/login",async (req,res)=>{

    try{

        const user=await User.findOne({email:req.body.email})

        if(!user){

            return res.status(404).json("User not found!")

        }

        const match=await bcrypt.compare(req.body.password,user.password)

        if(!match){

            return res.status(401).json("Wrong credentials!")

        }

        const token=jwt.sign({\_id:user.\_id,username:user.username,email:user.email},process.env.SECRET,{expiresIn:"3d"})

        const {password,...info}=user.\_doc

        res.cookie("token",token).status(200).json(info)

    }

    catch(err){

        res.status(500).json(err)

    }

})

//LOGOUT

router.get("/logout",async (req,res)=>{

    try{

        res.clearCookie("token",{sameSite:"none",secure:true}).status(200).send("User logged out successfully!")

    }

    catch(err){

        res.status(500).json(err)

    }

})

//REFETCH USER

router.get("/refetch", (req,res)=>{

    const token=req.cookies.token

    jwt.verify(token,process.env.SECRET,{},async (err,data)=>{

        if(err){

            return res.status(404).json(err)

        }

        res.status(200).json(data)

    })

})

module.exports=router

**Comments.js:**

const express=require('express')

const router=express.Router()

const User=require('../models/User')

const bcrypt=require('bcrypt')

const Post=require('../models/Post')

const Comment=require('../models/Comment')

const verifyToken = require('../verifyToken')

//CREATE

router.post("/create",verifyToken,async (req,res)=>{

    try{

        const newComment=new Comment(req.body)

        const savedComment=await newComment.save()

        res.status(200).json(savedComment)

    }

    catch(err){

        res.status(500).json(err)

    }

})

//UPDATE

router.put("/:id",verifyToken,async (req,res)=>{

    try{

        const updatedComment=await Comment.findByIdAndUpdate(req.params.id,{$set:req.body},{new:true})

        res.status(200).json(updatedComment)

    }

    catch(err){

        res.status(500).json(err)

    }

})

//DELETE

router.delete("/:id",verifyToken,async (req,res)=>{

    try{

        await Comment.findByIdAndDelete(req.params.id)

        res.status(200).json("Comment has been deleted!")

    }

    catch(err){

        res.status(500).json(err)

    }

})

//GET POST COMMENTS

router.get("/post/:postId",async (req,res)=>{

    try{

        const comments=await Comment.find({postId:req.params.postId})

        res.status(200).json(comments)

    }

    catch(err){

        res.status(500).json(err)

    }

})

module.exports=router

**Post.js:**

const express=require('express')

const router=express.Router()

const User=require('../models/User')

const bcrypt=require('bcrypt')

const Post=require('../models/Post')

const Comment=require('../models/Comment')

const verifyToken = require('../verifyToken')

//CREATE

router.post("/create",verifyToken,async (req,res)=>{

    try{

        const newPost=new Post(req.body)

        // console.log(req.body)

        const savedPost=await newPost.save()

        res.status(200).json(savedPost)

    }

    catch(err){

        res.status(500).json(err)

    }

})

//UPDATE

router.put("/:id",verifyToken,async (req,res)=>{

    try{

        const updatedPost=await Post.findByIdAndUpdate(req.params.id,{$set:req.body},{new:true})

        res.status(200).json(updatedPost)

    }

    catch(err){

        res.status(500).json(err)

    }

})

//DELETE

router.delete("/:id",verifyToken,async (req,res)=>{

    try{

        await Post.findByIdAndDelete(req.params.id)

        await Comment.deleteMany({postId:req.params.id})

        res.status(200).json("Post has been deleted!")

    }

    catch(err){

        res.status(500).json(err)

    }

})

//GET POST DETAILS

router.get("/:id",async (req,res)=>{

    try{

        const post=await Post.findById(req.params.id)

        res.status(200).json(post)

    }

    catch(err){

        res.status(500).json(err)

    }

})

//GET POSTS

router.get("/",async (req,res)=>{

    const query=req.query

    try{

        const searchFilter={

            title:{$regex:query.search, $options:"i"}

        }

        const posts=await Post.find(query.search?searchFilter:null)

        res.status(200).json(posts)

    }

    catch(err){

        res.status(500).json(err)

    }

})

//GET USER POSTS

router.get("/user/:userId",async (req,res)=>{

    try{

        const posts=await Post.find({userId:req.params.userId})

        res.status(200).json(posts)

    }

    catch(err){

        res.status(500).json(err)

    }

})

module.exports=router

**User.js:**

const express=require('express')

const router=express.Router()

const User=require('../models/User')

const bcrypt=require('bcrypt')

const Post=require('../models/Post')

const Comment=require('../models/Comment')

const verifyToken = require('../verifyToken')

//UPDATE

router.put("/:id",verifyToken,async (req,res)=>{

    try{

        if(req.body.password){

            const salt=await bcrypt.genSalt(10)

            req.body.password=await bcrypt.hashSync(req.body.password,salt)

        }

        const updatedUser=await User.findByIdAndUpdate(req.params.id,{$set:req.body},{new:true})

        res.status(200).json(updatedUser)

    }

    catch(err){

        res.status(500).json(err)

    }

})

//DELETE

router.delete("/:id",verifyToken,async (req,res)=>{

    try{

        await User.findByIdAndDelete(req.params.id)

        await Post.deleteMany({userId:req.params.id})

        await Comment.deleteMany({userId:req.params.id})

        res.status(200).json("User has been deleted!")

    }

    catch(err){

        res.status(500).json(err)

    }

})

//GET USER

router.get("/:id",async (req,res)=>{

    try{

        const user=await User.findById(req.params.id)

        const {password,...info}=user.\_doc

        res.status(200).json(info)

    }

    catch(err){

        res.status(500).json(err)

    }

})

module.exports=router

# FurtherImprovement

**Incorporate News Summary Feature:** Develop a feature that provides concise summaries of news articles, allowing users to quickly grasp key information.

**Integrate Sentiment Analysis:** Implement sentiment analysis to assess the emotional tone of news articles, providing users with insights into the overall sentiment of the content.

These improvements can enhance the website's performance, accessibility, and user satisfaction, ensuring that it remains responsive and functional for a broad range of users.

# Conclusion

In conclusion, the creation of the **‘News Times’** website involved a systematic and interconnected approach. Each phase of development was thoughtfully designed, with interdependencies and connections between them. This organizational structure streamlined the development process and ensured the successful creation of the news website.

For future enhancements, the addition of a caching mechanism is recommended, as it has the potential to significantly improve response times. This optimization will further enhance the user experience and make 'News Times' an even more efficient platform for accessing and exploring news content.

# Bibliography

1. Mozilla Developer Network(MDN)
2. HTML, CSS, and JavaScript Resources – w3schools.com
3. React.js
4. Express.js, Node.js, MongoDB
5. Visual Studio Code Documentation