LIST OF SYMBOLS / ABBREVIATIONS USED

CRUD:	Create, Read, Update, Delete
API:	Application Programming Interface
JWT:	JSON Web Token
UI:	User Interface
UX:	User Experience
DB:	Database

LIST OF FIGURES

CHAPTER 1 : INTRODUCTION

1.1 Overview of Internship

This report outlines the work undertaken during my internship at CodSoft, where I developed two projects as part of the MERN (MongoDB, Express, React, Node.js) stack: a blog posting application and a task management tool. The primary goal of this internship was to gain hands-on experience with full-stack development and understand the complete project lifecycle from planning to deployment.

1.2 Objectives

- To develop functional, user-friendly web applications.
- To implement and understand the MERN stack architecture.
- To enhance my skills in backend development using Express and Mongoose.
- To learn about project management and agile methodologies.

1.3 Scope of the Internship

The scope included requirement analysis, designing the architecture, coding, testing, and deploying the applications. Special emphasis was placed on implementing efficient code, error handling, and security measures in both projects.

CHAPTER 2: PROJECT 1 – BLOG POSTING APPLICATION

2.1 Overview

The Blog Posting Application is a web-based platform that allows users to create, edit, and manage blog posts. It is designed to enable smooth and interactive user experiences while handling content effectively.

2.2 Technology Stack

- **Frontend:** React.js, HTML, CSS, Tailwind
- **Backend**: Express.js, Node.js
- **Database:** MongoDB
- **Others**: RESTful APIs, JWT for authentication

2.3 Features

- **User Authentication:** Secure signup and login using JWT.
- **CRUD Operations**: Allows users to create, read, update, and delete blog posts.
- **Comments and Likes**: Users can engage with posts through comments and likes.
- **Admin Panel**: Special access for administrators to manage content and users**.**

2.4 Project Development

The development followed an agile methodology with iterative improvements. The architecture was divided into three main layers: frontend (React), backend (Node.js/Express), and database (MongoDB). RESTful APIs were used to communicate between the frontend and backend.

- **Frontend Development:** Implemented using React components for a dynamic and responsive UI.
- **Backend Development:** Developed APIs for handling user authentication, blog management, and data retrieval using Express.js.

- **Database Management**: MongoDB was used for data storage, providing scalability and flexibility for handling JSON-like documents**.**

2.5 Challenges and Solutions

- **Challenge**: Handling authentication and secure data access.
  Solution: Implemented JWT-based authentication to ensure secure user sessions.

- **Challenge:** Managing state across various components.
  Solution: Used React's Context API and hooks for effective state management.

2.6 Testing and Deployment

- Conducted unit and integration tests using Thunder-Client and Postman.

- Deployed the application on Render with MongoDB Atlas as the database service.

CHAPTER 3: PROJECT 2 – TASK MANAGEMENT TOOL

3.1 Overview

The Task Management Tool is designed to help users manage tasks, track progress, and organize work effectively. It offers user roles, task assignment, and real-time updates.

3.2 Technology Stack

- **Frontend:** React.js, Redux, Bootstrap
- **Backend:** Node.js, Express.js
- **Database**: MongoDB
- **Others:** RESTful APIs, JWT for authentication

3.3 Features

- **Task Creation and Assignment:** Users can create tasks and assign them to others**.**
- **Progress Tracking:** Status updates and progress bars to monitor task completion.
- **User Roles and Permissions:** Admins and users have different levels of access.
- **Notifications:** Alerts and reminders .

3.4 Project Development

The tool was developed using a component-based approach, with a focus on clean and modular code.

- **Frontend Development**: React and Redux were used to manage state and provide a seamless user experience.
- **Backend Development:** Built RESTful APIs for task management, role handling, and notifications.

- **Database Management**: Used MongoDB for its flexible schema design, allowing easy changes to data models as features expanded**.**

3.5 Challenges and Solutions

- **Challenge:** Implementing real-time updates for task status.
  Solution: Integrated WebSocket for real-time notifications and updates.
- **Challenge:** Managing complex state with multiple components**.**
  Solution: Employed Redux for state management to ensure a scalable and maintainable codebase.
-

3.6 Testing and Deployment

- Testing was done using Thunder-client and Post-man for end-to-end testing.
- Deployed using Render for the frontend and the backend, ensuring reliable performance.

CHAPTER 4: TECHNICAL CHALLENGES AND SOLUTIONS

During the development of both projects, several technical challenges were faced, including API integration, database optimization, and frontend responsiveness. Detailed analysis and step-by-step solutions to these challenges were documented to enhance future development processes.

CHAPTER 5: LEARNINGS AND IMPROVEMENTS

The internship provided significant learning opportunities, including:

- Enhanced understanding of full-stack development and agile methodologies.
- Improved debugging skills, especially in managing state and handling backend errors.
- Learned best practices in authentication, data validation, and error handling.

CHAPTER 6: CONCLUSION

The internship at CodSoft was a highly rewarding experience that allowed me to apply theoretical knowledge to practical problems. By developing two complete web applications, I gained valuable insights into the complexities of full-stack development. This experience has equipped me with the skills and confidence needed to tackle future challenges in the field of software development**.**

REFERENCES

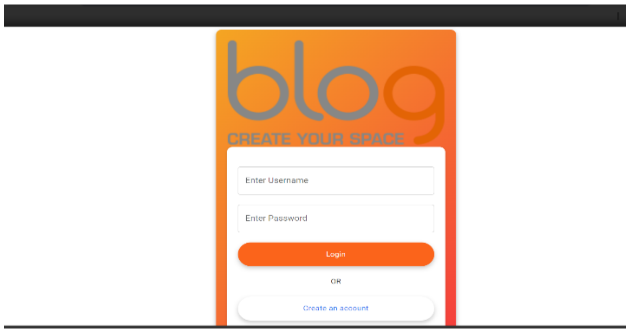
1. **Smith, J.,** "Building Scalable Web Applications," Journal of Web Development, Vol. 12, pp. 45-67, 2023.
2. **Doe,** A. and Lee, R., "MERN Stack Essentials," Software Engineering Review, Vol. 8, pp. 101-115, 2022.
3. **Johnson et al., "**React Best Practices," Frontend Developer Journal, Vol. 6, pp. 89-102, 2021.

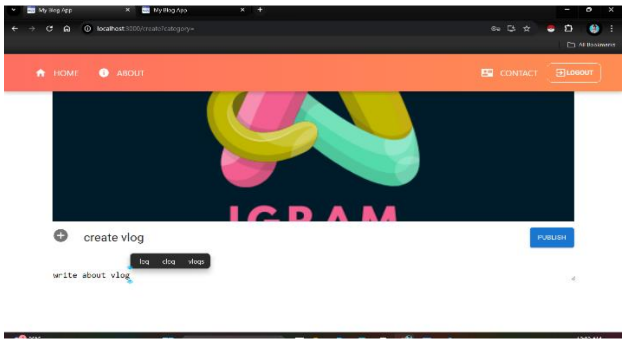
APPENDICES

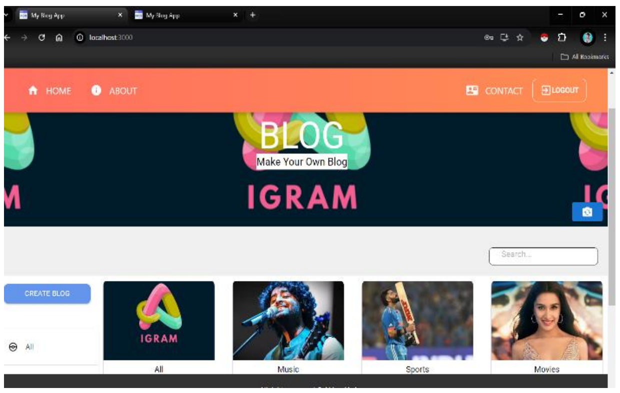
Appendix I: Project Screenshots

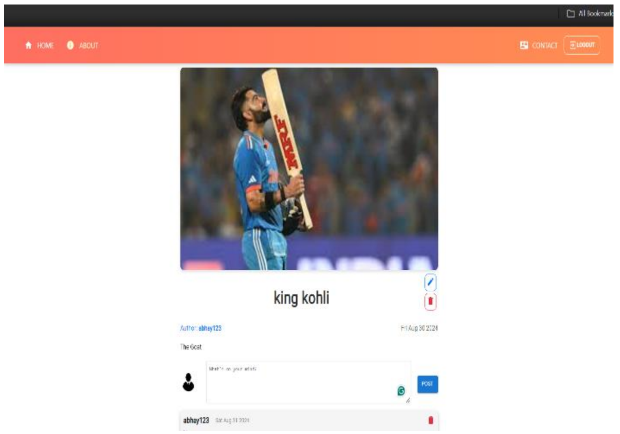
First project

Project link: https://codsoft-myblog-app1-0.onrender.com

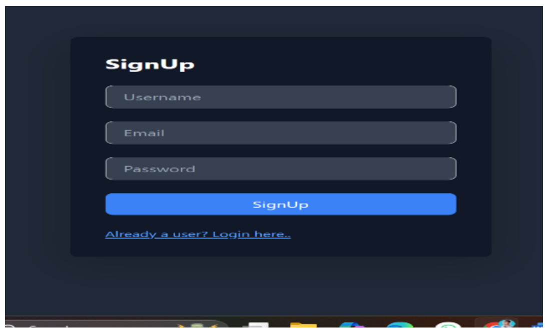


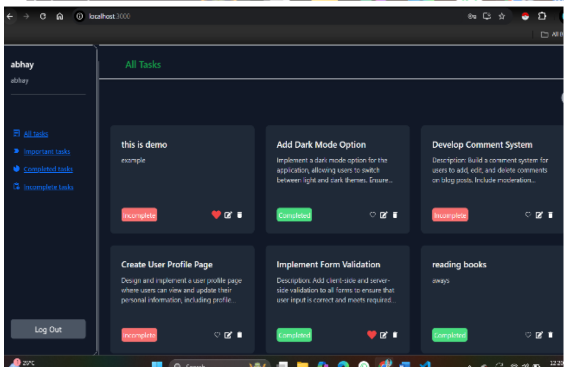






Second Project





this is demo
example
Incomplete

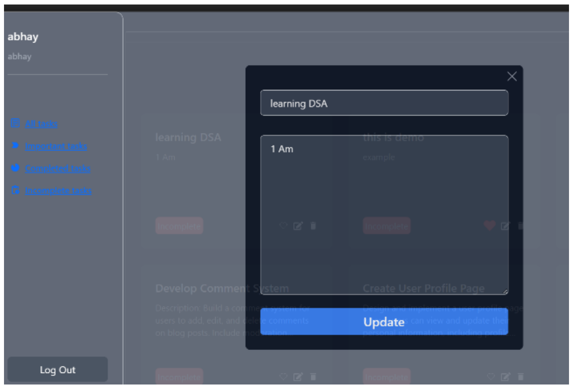
Add Dark Mode Option
Implement a dark mode option for the application, allowing users to switch between light and dark themes. Ensure...
Completed

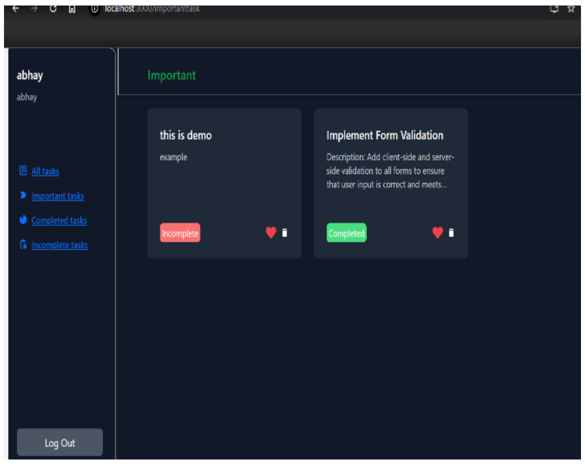
Develop Comment System
Description: Build a comment system for users to add, edit, and delete comments on blog posts. Include moderation...
Incomplete

Create User Profile Page
Design and implement a user profile page where users can view and update their personal information, including profile...
Incomplete

Implement Form Validation
Description: Add client-side and server-side validation to all forms to ensure that user input is correct and meets required...
Completed

reading books
aways
Completed



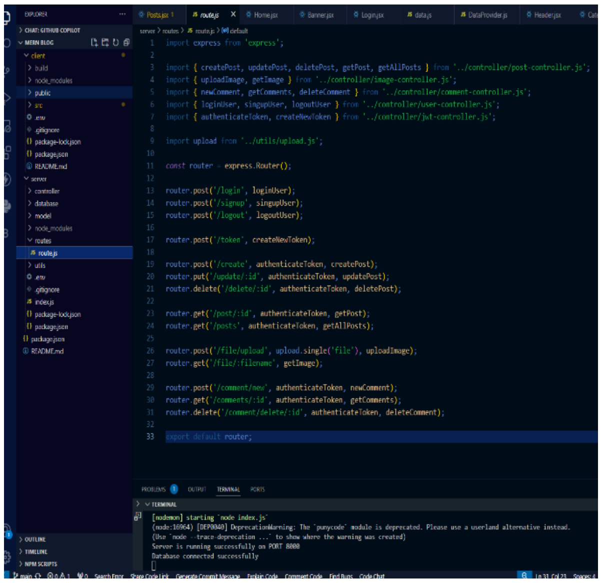


this is demo
example
Incomplete

Implement Form Validation
Description: Add client-side and server-side validation to all forms to ensure that user input is correct and meets...
Completed

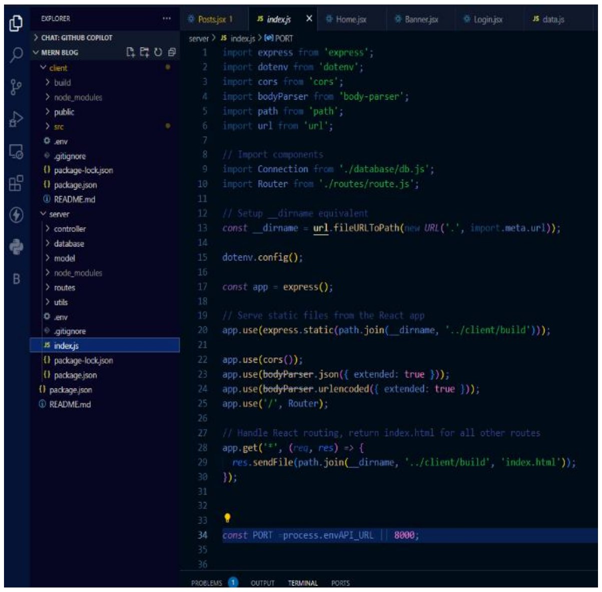
Appendix II: Code Snippets

Some Code of 1st project



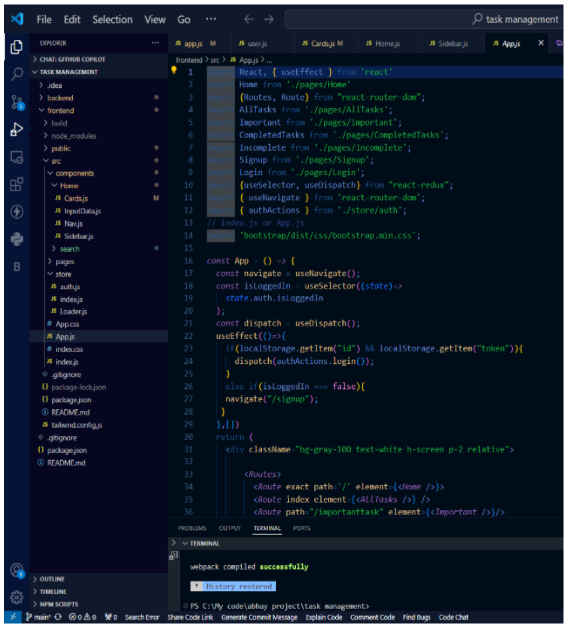
```
import express from 'express';

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;
```

```
[nodemon] starting `node index.js`
(node:16964) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Server is running successfully on PORT 8000
Database connected successfully
```



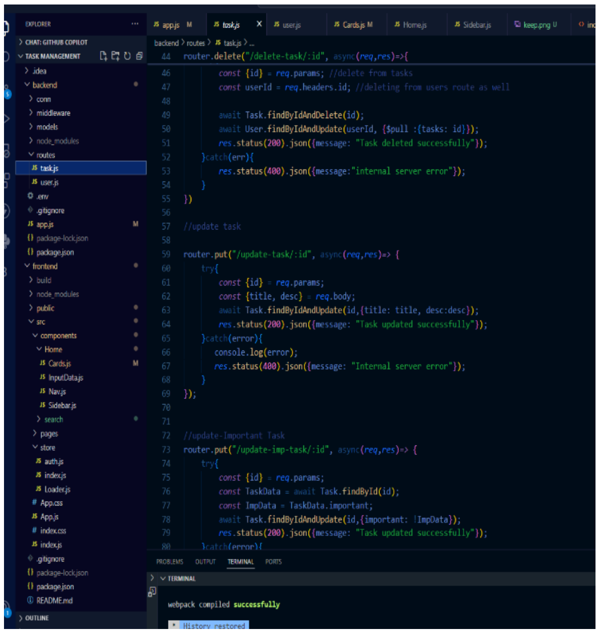
```
import express from 'express';
import dotenv from 'dotenv';
import cors from 'cors';
import bodyParser from 'body-parser';
import path from 'path';
import url from 'url';

// Import components
import Connection from './database/db.js';
import Router from './routes/route.js';

// Setup __dirname equivalent
const __dirname = url.fileURLToPath(new URL('.', import.meta.url));

dotenv.config();

const app = express();

// Serve static files from the React app
app.use(express.static(path.join(__dirname, '../client/build')));

app.use(cors());
app.use(bodyParser.json({ extended: true }));
app.use(bodyParser.urlencoded({ extended: true }));
app.use('/', Router);

// Handle React routing, return index.html for all other routes
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, '../client/build', 'index.html'));
});



const PORT =process.envAPI_URL || 8000;
```

Some code of 2nd project



```
import React, { useEffect } from 'react'
import Home from './pages/Home'
import {Routes, Route} from "react-router-dom";
import AllTasks from './pages/AllTasks';
import Important from './pages/Important';
import CompletedTasks from './pages/CompletedTasks';
import Incomplete from './pages/Incomplete';
import Signup from './pages/Signup';
import Login from './pages/Login';
import {useSelector, useDispatch} from "react-redux";
import { useNavigate } from 'react-router-dom';
import { authActions } from './store/auth';
// index.js or App.js
import 'bootstrap/dist/css/bootstrap.min.css';

const App = () => {
  const navigate = useNavigate();
  const isLoggedIn = useSelector((state)=>
    state.auth.isLoggedIn
  );
  const dispatch = useDispatch();
  useEffect(()=>{
    if(localStorage.getItem("id") && localStorage.getItem("token")){
      dispatch(authActions.login());
    }
    else if(isLoggedIn === false){
    navigate("/signup");
   }
  },[])
  return (
    <div className="bg-gray-100 text-white h-screen p-2 relative">

        <Routes>
          <Route exact path='/' element={<Home />}>
          <Route index element={<AllTasks />} />
          <Route path="/importanttask" element={<Important />}/>
```

```
webpack compiled successfully

 * History restored

PS C:\My code\abhay project\task management>
```



```
router.delete("/delete-task/:id", async(req,res)=>{
        const {id} = req.params; //delete from tasks
        const userId = req.headers.id; //deleting from users route as well

        await Task.findByIdAndDelete(id);
        await User.findByIdAndUpdate(userId, {$pull :{tasks: id}});
        res.status(200).json({message: "Task deleted successfully"});
    }catch(err){
        res.status(400).json({message:"internal server error"});
    }
})

//update task

router.put("/update-task/:id", async(req,res)=> {
    try{
        const {id} = req.params;
        const {title, desc} = req.body;
        await Task.findByIdAndUpdate(id,{title: title, desc:desc});
        res.status(200).json({message: "Task updated successfully"});
    }catch(error){
       console.log(error);
       res.status(400).json({message: "Internal server error"});
    }
});


//update-Important Task
router.put("/update-imp-task/:id", async(req,res)=> {
    try{
        const {id} = req.params;
        const TaskData = await Task.findById(id);
        const ImpData = TaskData.important;
        await Task.findByIdAndUpdate(id,{important: !ImpData});
        res.status(200).json({message: "Task updated successfully"});
    }catch(error){
```

```
webpack compiled successfully

 * History restored
```