FULL STACK DEVELOPMENT USING MERN

CipherSchools

A Training Report

Submitted in Partial Fulfillment of the Requirements for the Award of the Degree of

B. Tech CSE

DATA SCIENCE

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB

From 06/05/2024 to 07/24/2024

SUBMITTED BY

Name of student: Sanvi Ojha

Registration Number: 12212400

Signature of the student: Sanvi Ojha

DECLARATION

To Whom so ever it may concern

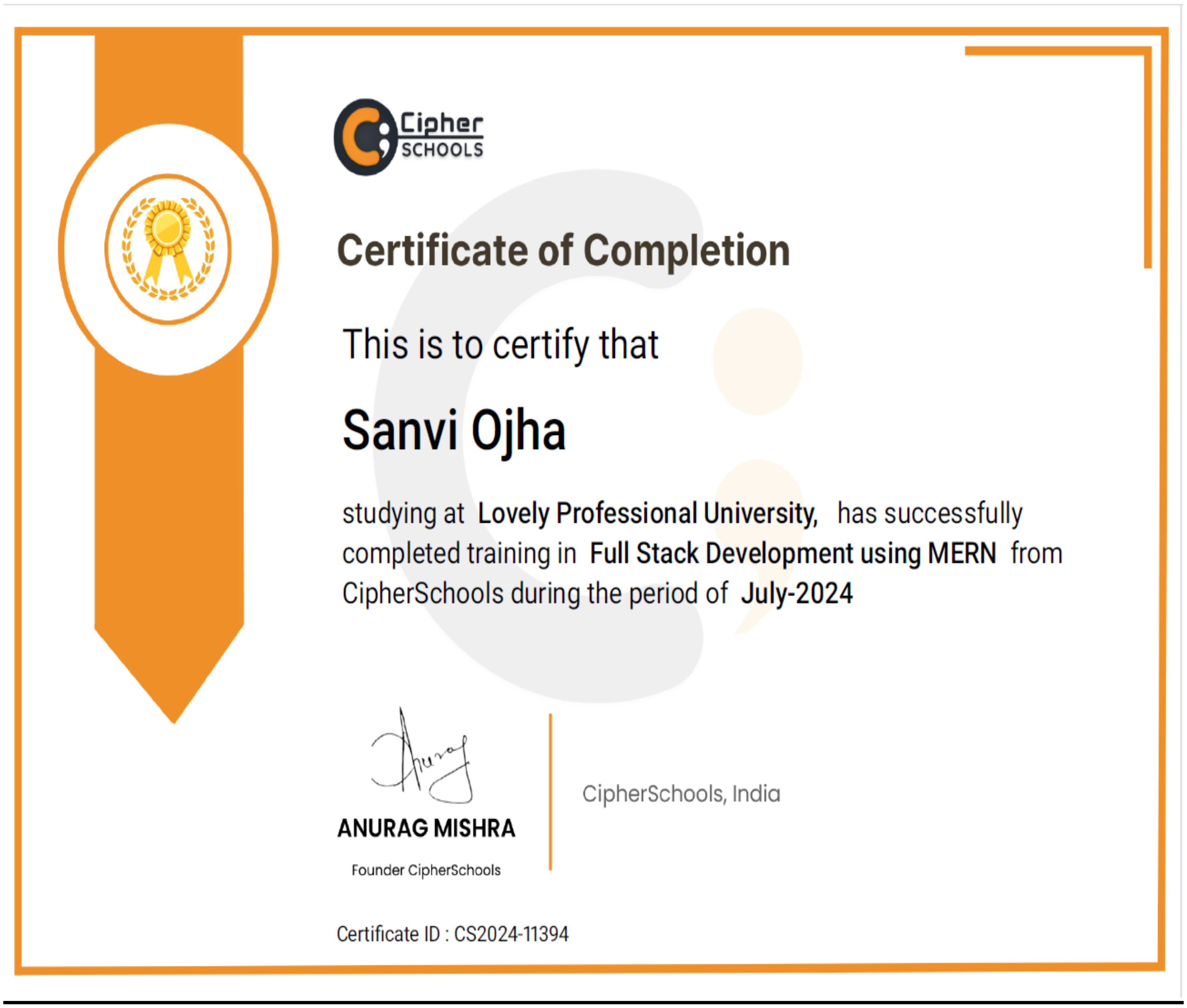
I, **Sanvi Ojha, Registration Number 12212400**, hereby declare that the work presented in this report titled "**Full Stack Development Using MERN**," completed between **June 2024 to July 2024**, is an original piece of work carried out by me. This report is submitted as part of the requirements for the **Bachelor of Technology in Computer Science & Engineering.**

Name of the Student: Sanvi Ojha

Registration Number: 12212400

Date: 31/08/2024

TRAINING CERTIFICATE



Cipher SCHOOLS

Certificate of Completion

This is to certify that

Sanvi Ojha

studying at **Lovely Professional University,** has successfully completed training in **Full Stack Development using MERN** from CipherSchools during the period of **July-2024**

[illegible]

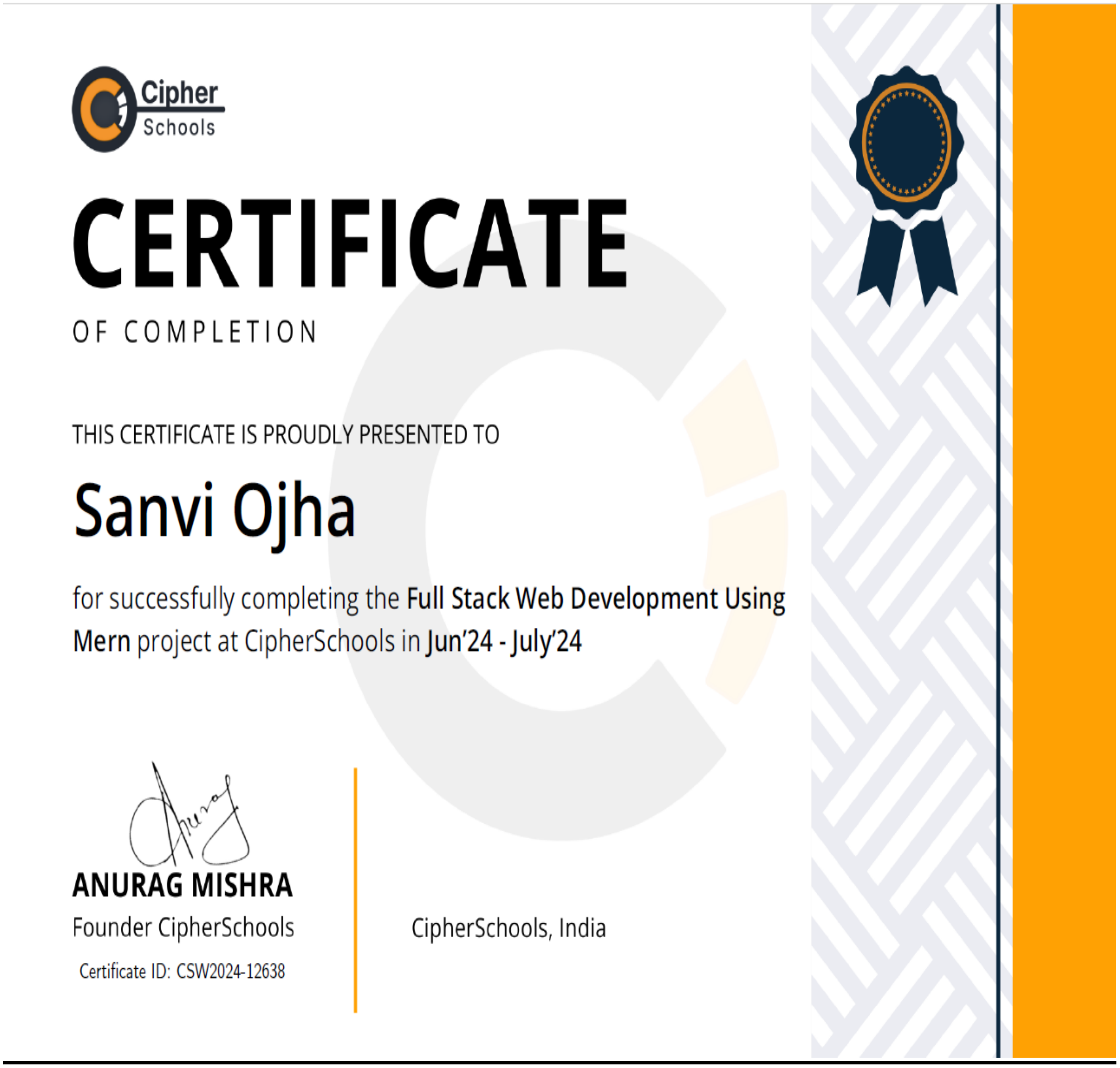
ANURAG MISHRA

Founder CipherSchools

CipherSchools, India

Certificate ID : CS2024-11394

PROJECT CERTIFICATE



Cipher Schools

CERTIFICATE

OF COMPLETION

THIS CERTIFICATE IS PROUDLY PRESENTED TO

Sanvi Ojha

for successfully completing the **Full Stack Web Development Using Mern** project at CipherSchools in **Jun'24 - July'24**

[illegible]

ANURAG MISHRA

Founder CipherSchools

Certificate ID: CSW2024-12638

CipherSchools, India

INTRODUCTION OF THE PROJECT UNDERTAKEN

The project involved an in-depth exploration of Full Stack Development using the MERN (MongoDB, Express.js, React.js, Node.js) stack. The work included the creation of three smaller projects and one comprehensive summer training project. The mini-projects comprised a personal resume website, a Tic-Tac-Toe game, and a Notes Command-Line Interface (CLI) tool. The summer training project was a Blogging website, which brought together all the skills and knowledge developed during the training.

- **Objectives of the Work Undertaken**

The primary objectives of this project were to:

1. Develop Technical Competence: To gain hands-on experience with the MERN stack by working on practical web development projects.

2. Apply Learning Practically: To implement theoretical knowledge into real-world applications that address specific challenges.

3. Enhance Problem-Solving Skills: To tackle and overcome the technical challenges encountered during development, improving analytical and troubleshooting abilities.

4. Create a Professional Portfolio: To build a portfolio showcasing full-stack development skills to potential employers or clients.

- **Scope of the Work**

The scope of this work included:

1. Frontend Development: Crafting user interfaces using React.js, emphasizing responsive design, reusable components, and delivering intuitive user experience.

2. Backend Development: Building server-side logic with Node.js and Express.js, ensuring effective communication between the frontend and backend.

3. Database Management: Designing and managing database structures using MongoDB, focusing on efficient data storage, retrieval, and manipulation.

4. Deployment and Testing: Deploying the applications into a live environment and performing thorough testing to ensure they are secure, performant, and reliable.

- **Importance and Applicability**
  This project is particularly significant as it provided practical experience in building web applications with the MERN stack, illustrating the ability to integrate various technologies into a cohesive solution. The mini projects helped build core skills, while the Blogging website, like the summer training project, allowed for the application of these skills in a more complex, real-world scenario.
- **Key aspects of the project include:**

1. Integration of Technologies: Demonstrating the effective combination of frontend and backend technologies to create a seamless user experience.

2. User Experience Design: Emphasizing the importance of designing user-friendly, visually appealing interfaces.

3. Data Handling: Highlighting the role of effective database management in managing large volumes of data.

4. Problem-Solving in Development: Addressing common challenges in web development, including state management, API integration, and asynchronous operations.

- **Role and Profile**

  In this project, I learned and implemented full stack knowledge which I gained and, with responsibilities that included:

1. Project Planning: Defining the project scope, setting objectives, and planning the development timeline.

2. Frontend Development: Designing and implementing user interfaces for each project, with a focus on usability and responsiveness.

3. Backend Development: Creating server-side logic, managing database interactions, and building APIs.

4. Testing and Debugging: Conducting comprehensive testing to identify and resolve bugs, ensuring that the applications work as intended.

5. Deployment: Deploying the applications to a live environment, ensuring that they are accessible to users.

In this role, I utilized my expertise (which I learned) in HTML, CSS, JavaScript, React.js, Node.js, Express.js, and MongoDB to develop functional, user-focused web applications. I also adhered to best practices in coding, version control, and project management to deliver high-quality, professional work.

INTRODUCTION OF THE COMPANY

Company's vision & mission:

CipherSchools is an educational video-streaming company based in India. We are the ultimate destination for Content Creators and Students who crave an exhilarating online learning experience. Our platform is designed to ignite your Imagination, unleash your potential, and revolutionize the way you create, share, and learn. Since our inception in 2020, we have left an indelible mark on countless students around the globe, transforming their lives and empowering them to conquer new horizons.

Join us on this thrilling journey and be a part of our incredible success story.

Our Vision:

Envision a world where high-quality online learning is readily accessible to everyone, regardless of their location. At our core, we aspire to transform this vision into a tangible reality by becoming the go-to platform for anyone seeking free online learning experiences.

Our Mission:

We are on a mission to bridge the gap between passionate, unskilled students and seasoned industry experts. By connecting these two groups, we aim to empower students and help them realize their career aspirations.

Origin and growth of company: Refer: https://www.linkedin.com/company/cipherschool/

Various departments and their functions:

1. Technical department
2. Operations department
3. Marketing department

Organization chart of the company:

1. CEO & Founder: Mr. Anurag Mishra
2. Technical Department: Headed by Nitesh Kumar
3. Operations Department: Headed by Geetika
4. Marketing Team: Headed by Sanskar

Address of the company:

Chandigarh Citi Center (CCC), VIP Road, Punjab - 140603

Brief Description of the Work Done

During my summer training in Full Stack Development using the MERN stack, I encountered initial difficulties due to my data science background, particularly in grasping the syntax and workflows of web development technologies. However, with consistent effort, I gradually learned the necessary skills and successfully completed the project. This training has significantly broadened my technical abilities, particularly in web development, and will be invaluable for my upcoming placement projects and future career endeavors.

- **Position During Summer Training and Roles**

In this summer training, I served as a Full Stack Developer, with responsibilities that included:

1. Learning and Application: Bridging my data science background with full-stack development by learning new technologies and applying them to practical projects.

2. Frontend Development: Crafting responsive, user-friendly interfaces using React.js, HTML, CSS, and JavaScript.

3. Backend Development: Creating server-side logic with Node.js and Express.js, managing data interactions, and ensuring smooth communication between the frontend and backend.

4. Database Management: Utilizing MongoDB to design and manage databases, focusing on efficient data storage and retrieval.

5. Testing and Debugging: Conducting comprehensive testing and debugging to ensure the functionality and reliability of the applications.

- **Activities/Equipment Handled**

Throughout the training, I engaged with a range of technologies and tools, each with its own uses, advantages, and disadvantages:

1. <u>HTML (Hypertext Markup Language):</u> HTML is the standard markup language for creating web pages. It structures the content on the web.

2. <u>CSS (Cascading Style Sheets):</u>CSS is used to style and layout web pages, allowing for customization of colors, fonts, and overall design.

3. <u>JavaScript:</u> JavaScript is a programming language used to create dynamic and interactive content on websites, such as animations, form validation, and API integration.

4. <u>React.js:</u>

React is a popular open-source JavaScript library developed by Facebook for building user interfaces, specifically for single-page applications where you want a fast, interactive user experience. React allows developers to create large web applications that can update and render efficiently in response to data changes. It uses component-based architecture, enabling developers to build encapsulated components that manage their own state and then compose them to create complex UIs. The core idea of React is to build UIs using components.

Key Concepts in React:
Props:

- Props (short for "properties") are used to pass data from one component to another.
- Props are immutable, meaning they cannot be changed by the component that receives them.

State:

- State is used to manage data that changes over time within a component.
- Unlike props, state is mutable and can be updated by the component itself.
- State updates trigger re-rendering of the component to reflect the changes in the UI.

Event Handling:

- React allows you to handle events (like clicks, form submissions) using event handlers.
- Event handlers are written in camelCase, and you pass them as props to elements.
- You can respond to events by declaring event handler functions inside your components:

Applications/Uses of React:

1. **Creating Interactive User Interfaces:**
   - React allows developers to create highly interactive and dynamic UIs. It handles UI updates efficiently by using a virtual DOM, ensuring that only the necessary parts of the page are re-rendered when the state changes.
2. **Developing Reusable Components:**
   - Reacts component-based architecture enables the creation of reusable UI components. These components can be used across different parts of an application or even in different projects, promoting code reusability and consistency.
3. **Building Mobile Applications (React Native):**
   - React is the foundation for React Native, a framework that allows developers to build mobile applications for iOS and Android using React. This allows for code sharing between web and mobile applications, speeding up development.

How React is faster than other libraries and frameworks?

1. **Virtual DOM: Virtual DOM** and **diffing algorithm** reduce the cost of direct DOM manipulation.

2. **Efficient Reconciliation: Efficient reconciliation** with the use of keys helps manage dynamic lists.

3. **Component-Based Architecture: Component-based architecture** promotes modular and reusable code.

4. **Asynchronous Rendering (Concurrent Mode): Concurrent Mode** and **Suspense** improve responsiveness and loading times.

5. **Optimized Updates with Hooks: Hooks** enable performance optimizations by memoizing calculations and functions.

5. Node.js:

- Uses: Node.js is a runtime environment that allows JavaScript to be used for backend development, enabling server-side scripting.

- Advantages: High performance for I/O-bound tasks, large ecosystem (npm), enables full-stack development with a single language.

- Disadvantages: Not suited for CPU-intensive tasks, callback hell (though mitigated by Promises and async/await), can be complex to manage in large applications.

- Future Scope: Node.js will continue to grow in popularity, especially with the increasing demand for scalable, real-time applications.

6. MongoDB:

- Uses: MongoDB is a NoSQL database that stores data in flexible, JSON-like documents, making it ideal for applications with unstructured data.

- Advantages: Flexibility in data storage, scalability, easy integration with JavaScript-based frameworks.

- Disadvantages: Lack of complex transactions, potential issues with data consistency, requires learning new querying syntax (MongoDB Query Language).

- Future Scope: MongoDB will continue to be a strong choice for modern web applications, particularly those requiring flexible and scalable data storage solutions.

7. API (Application Programming Interface):

- Uses: APIs allow different software systems to communicate with each other, enabling data exchange and functionality sharing.

- Advantages: Facilitates integration between different systems, promotes modular development, enhances functionality by connecting to external services.

- Disadvantages: Can introduce security risks if not properly managed, dependency on third-party services can lead to reliability issues, require careful version management.

- Future Scope: APIs will remain central to software development, with growing importance in microservices architecture and cloud-based applications.

- **Challenges Faced and How They Were Tackled**

1. **Understanding Difficult Topics:** Faced challenges with complex topics, which required revisiting them through YouTube tutorials and class recordings multiple times to gain a clear understanding.
2. **Balancing Multiple Projects:** Managed the complexity of working on both intermediate and major projects simultaneously, which required effective time management and prioritization.
3. **Debugging Complex Issues:** Encountering and resolving bugs was more challenging due to the interconnected nature of full-stack applications.
4. **Integrating Technologies:** Faced challenges integrating different technologies within the MERN stack, requiring additional research and experimentation to ensure seamless functionality.
5. **Deployment Difficulties:** Experienced issues during the deployment phase, such as configuring environment variables and ensuring compatibility between the development and production environments. Overcame these challenges by researching deployment best practices and seeking guidance from resources and mentors.

- **Learning Outcomes**

- **Understand the MERN Stack:** Learn how to use MongoDB, Express.js, React.js, and Node.js together to build complete web applications.
- **Create Reusable Parts in React:** Learn how to build small, reusable pieces (components) that can be combined to create user-friendly web interfaces.
- **Build APIs with Express.js:** Understand how to create and use APIs to connect the front end of a website with the back end, allowing different parts of the application to communicate.
- **Manage Data with MongoDB:** Learn how to set up and work with databases to store, retrieve, and manage data effectively.
- **Combine Front-End and Back-End:** Understand how to bring together the visual parts of a website (front-end) with the behind-the-scenes logic (back-end) to create a full, working web application.
- **Implement User Login and Security:** Learn how to set up user accounts and secure their information, making sure only authorized users can access certain features.
- **Deploy Applications Online:** Gain experience in putting your projects on the internet so that they can be accessed by others.
- **Use Git for Version Control:** Improve your ability to track changes, collaborate with others, and manage different versions of your projects using Git.
- **Gain Industry Knowledge:** Understand important practices in the IT industry, including how to manage projects, work well in teams, and maintain a healthy work-life balance.

- **Data Analysis**

While the focus was on full-stack development, my background in data science was instrumental in understanding and managing data effectively. I applied my data analysis skills to design efficient database schemas and optimize data retrieval processes. This analytical mindset also helped in debugging and optimizing the performance of the applications, ensuring they were both functional and efficient.

The combination of data science and full-stack development skills acquired during this training positions me well for future opportunities, allowing me to contribute to a wide range of projects and roles.

PROJECTS

Resume

The 'my-portfolio' project is a React-based single-page application designed to showcase my personal and professional profile. This project serves as an online portfolio, enabling visitors to navigate through various sections that highlight different aspects of my background and skills.

The application features a user-friendly navigation bar that allows seamless movement between the key sections of the webpage. The **header** introduces the website, followed by the **About** section, which provides a brief overview of my personal and professional background. The **Tools** section details the technology stack I am proficient in, offering insights into the tools and frameworks used to develop this portfolio.

About Section:

The **About** section provides a concise overview of my background, interests, and aspirations. It highlights my journey into the world of technology, my passion for coding, and my commitment to continuous learning and growth in the field of software development. This section is designed to give visitors a snapshot of who I am, both personally and professionally.

Tech Stack:

In the **Tools** section, I have listed the key technologies and programming languages I am proficient in. My tech stack includes:

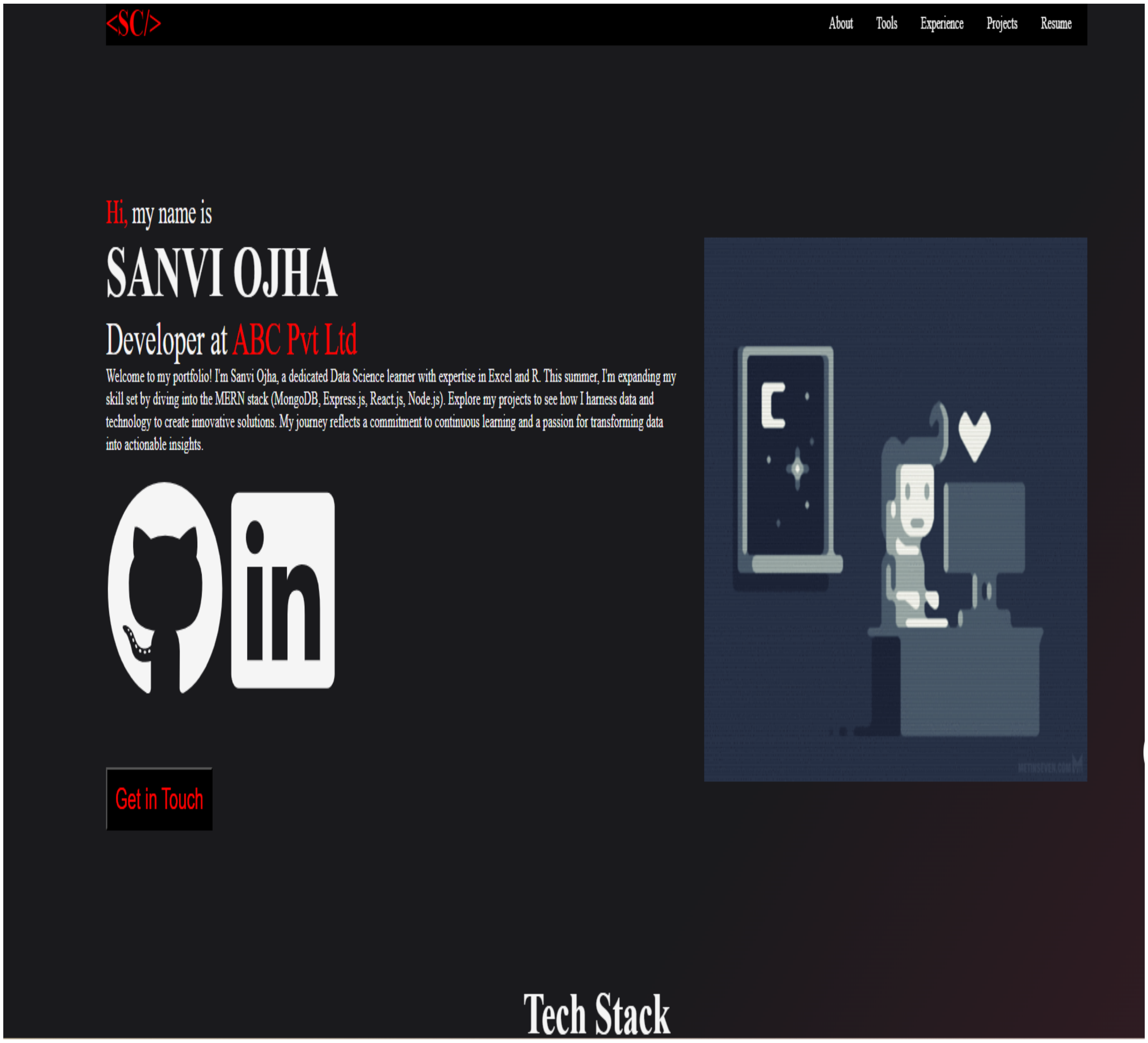
- **HTML** and **CSS** for building and styling web pages, ensuring they are both visually appealing and responsive.
- **JavaScript** for adding interactivity and dynamic content to web applications.
- **C++**, **C**, and **Python** for general-purpose programming, algorithm development, and problem-solving, showcasing my versatility in both low-level and high-level programming languages.

This diverse set of tools reflects my ability to work across various aspects of software development, from frontend design to backend logic and algorithm implementation.

Overall, the 'my-portfolio' project is a comprehensive digital resume that effectively communicates my technical skills and project experience in an organized and accessible format.

Live Link - https://sanviojharesume09.netlify.app/

Screenshots –



Hi, my name is

SANVI OJHA

Developer at ABC Pvt Ltd

Welcome to my portfolio! I'm Sanvi Ojha, a dedicated Data Science learner with expertise in Excel and R. This summer, I'm expanding my skill set by diving into the MERN stack (MongoDB, Express.js, React.js, Node.js). Explore my projects to see how I harness data and technology to create innovative solutions. My journey reflects a commitment to continuous learning and a passion for transforming data into actionable insights.

Tech Stack

Key Takeaways from The Project:

Commands Used:

1. `npm create vite`:
   The **npm create vite@latest** command is used to create a new project using Vite.
   npm: The Node Package Manager command used to run scripts and manage dependencies.
   create: A convention for creating a new project or running a generator script.
   vite: The specific generator script for creating a Vite project.
   @latest: Ensures that the latest version of the create-vite package is used.
   What is Vite?
   Vite is a modern front-end build tool that offers a faster and leaner development experience for modern web projects.

2. Once the project is created, navigate into the project directory:
   `cd project_name`

3. After navigating into the project directory, install the necessary dependencies:
   `npm install`

4. To start the development server and see your Vite-powered application in action:
   npm run dev

5. To create an optimized production build of your application:
   npm run build

What is use of **assets** folder?

In a Vite project, the assets folder is typically used to store static assets such as images, fonts, icons, and other files that your application needs to reference directly. These assets are included in your project and can be referenced in your code, for example, in your HTML, CSS, or JavaScript files.

Key Points about the assets Folder in Vite:

1) Default Location for Static Assets:

By convention, the assets folder is where you place your static files that are not processed by JavaScript. This includes images like .jpg, .png, fonts like .woff, .ttf, and other files you might want to include in your project.

2) Relative Imports:

You can reference the files in the assets folder using relative paths in your components.

```
import logo from './assets/logo.png';



function App() {
```

```
return (
  <div>
    <img src={logo} alt="Logo" />
  </div>
 );
}
```

3) Processed by Vite:
Vite handles static assets intelligently. During development, it serves the assets directly. When building for production, Vite processes the assets to optimize them (e.g., compressing images) and places them in the dist folder with hashed filenames for cache busting.

Example of a processed asset URL in production:

```
<img src="/assets/logo.8d7412b.png" alt="Logo">
```

4) Using Public Folder for Non-Processed Assets:
If you want to include assets that should not be processed by Vite (for instance, if you need them to remain at their original paths), you can use the public folder. Assets placed in the public folder are copied as-is to the root of the dist directory.

Example:

File: public/favicon.ico

Usage: <link rel="icon" href="/favicon.ico" />

What is **node_modules** folder in react?

package.json:

package-lock.json :

The package-lock.json file is automatically generated by npm when you run npm install. It ensures that the project's dependencies are installed exactly as they were when the file was created, providing a consistent environment across different machines.

Key Contents:

1. Exact Versions:
Records the exact versions of each installed package, including nested dependencies, ensuring reproducibility.

2. Dependency Tree:
Lists all dependencies and sub-dependencies, along with their exact versions, download URLs, and integrity checksums. This ensures that everyone working on the project has the exact same dependency tree.

3. Locking Versions:

Even if package.json specifies a range (e.g., "react": "^18.0.0"), package-lock.json will lock it to the specific version installed (e.g., "react": "18.0.0").

4. Primary Function:

package-lock.json ensures consistency across different environments by locking the versions of dependencies and sub-dependencies. This prevents the "works on my machine" problem, where the same project behaves differently on different setups due to slightly different dependency versions.

5. Automatic Generation and Maintenance:

This file is automatically generated and should not be manually edited. It updates automatically whenever you run npm install, npm update, or change dependencies in package.json.

6. Not Meant for Manual Edits:

Unlike package.json, developers typically do not manually edit package-lock.json. Any changes are handled by npm itself.

Tic-Tac Toe Project

This project is a web-based implementation of the classic Tic-Tac-Toe game, developed using React, a popular JavaScript library for building user interfaces. The game allows two players to take turns placing their marks (X or O) on a 3x3 grid. The objective is to be the first player to align three marks horizontally, vertically, or diagonally.

The application is designed with a clean and intuitive user interface, ensuring a seamless gaming experience. It features a history of moves, allowing players to navigate through previous game states and even jump back to earlier moves.

Below is an explanation of the logic behind each part of the code.

1) Imports and Initial Setup:

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

import "./App.css";
```

The code imports the useState hook from React, which is used for managing state within functional components.

The App.css file is imported for styling the components.

2) calculateWinner Function:

```
function calculateWinner(squares) {

  const winningCombinations = [

    [0, 1, 2],

    [3, 4, 5],
    [6, 7, 8],

    [0, 3, 6],

    [1, 4, 7],

    [2, 5, 8],

    [0, 4, 8],

    [2, 4, 6],

  ];

  for (let i = 0; i < winningCombinations.length; i++) {

    const [a, b, c] = winningCombinations[i];

    if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {

     return squares[a];

    } } return null;  }
```

This function checks if there is a winner. It takes the squares array as input, which represents the current state of the game board.The winning Combinations array contains all possible winning combinations for the Tic-Tac-Toe game.The function loops through each winning combination and checks if the values at the indices of the combination are equal and not null. If they are, it returns the value (X or O) as the winner.If no winner is found, the function returns null.

3) Square Component:

```
function Square({ value, onSquareClick }) {

  return (

   <button onClick={onSquareClick} className="square">

    {value}

   </button>

  ); }
```

The Square component represents a single square on the Tic-Tac-Toe board.

It receives two props: value (the current value of the square, either X, O, or null) and onSquareClick (a function to handle click events).

The square is rendered as a button, and clicking the button triggers the onSquareClick function.

4) TicTacToe Component:

```
function TicTacToe() {
  const [xisNext, setXisNext] = useState(true);
  const [history, setHistory] = useState([Array(9).fill(null)]);
  const [currentMove, setCurrentMove] = useState(0);
  const currentSquares = history[currentMove];
  function goTo(move) {
    setCurrentMove(move);
    setXisNext(move % 2 === 0);
  }
 function handleHistory(squares) {
    const newHistory = [...history.slice(0, currentMove + 1), squares];
    setHistory(newHistory);
    setCurrentMove(newHistory.length - 1);
    setXisNext(!xisNext);
  }
  const moves = history.map((squares, move) => {
    let desc = move > 0 ? `Go to move : #${move}` : "Go to starting of the game";
    return (
      <li key={move}>
        <button onClick={() => goTo(move)}>{desc}</button>
      </li>
    );
  });

  return (
    <div className="game">
      <div className="game-board">
        <Board
          xisNext={xisNext}
          squares={currentSquares}
          handleHistory={handleHistory}
        />
      </div>
      <div className="game-info">
        <ol>{moves}</ol>
      </div>
    </div>
  );
}
```

The TicTacToe component manages the state of the game and coordinates between the Board and Square components.

State Variables:

xisNext: A boolean that determines if the next player is X (true) or O (false).

history: An array that stores the history of moves as arrays of 9 squares.

currentMove: Tracks the index of the current move in the history.

currentSquares: Represents the squares array of the current move.

Functions:

goTo: Jumps to a specific move in the game's history, updating currentMove and determining which player is next based on the move index.

handleHistory: Updates the game history when a move is made, adds the new state of the board to the history, and updates the current move.

Rendering:

The game board is rendered using the Board component.

The game history is rendered as a list of buttons, allowing players to jump to previous moves.

5. Board Component:

```
function Board({ xisNext, squares, handleHistory }) {

  function handleClick(i) {

    if (squares[i] || calculateWinner(squares)) {

      return;

    }



    const updatedSquares = squares.slice();

    updatedSquares[i] = xisNext ? "X" : "O";

    handleHistory(updatedSquares);

  }
```

```
const winner = calculateWinner(squares);

let status = winner ? `Winner is : ${winner}` : `Next player is : ${xisNext ? "X" : "O"}`;


return (

 <>

  <div className="status">{status} </div>

  <div className="board-row">

   <Square value={squares[0]} onSquareClick={() => handleClick(0)} />

   <Square value={squares[1]} onSquareClick={() => handleClick(1)} />

   <Square value={squares[2]} onSquareClick={() => handleClick(2)} />

  </div>

  <div className="board-row">

   <Square value={squares[3]} onSquareClick={() => handleClick(3)} />

   <Square value={squares[4]} onSquareClick={() => handleClick(4)} />

   <Square value={squares[5]} onSquareClick={() => handleClick(5)} />

  </div>

  <div className="board-row">

   <Square value={squares[6]} onSquareClick={() => handleClick(6)} />

   <Square value={squares[7]} onSquareClick={() => handleClick(7)} />

   <Square value={squares[8]} onSquareClick={() => handleClick(8)} />

  </div>

 </>
```

```
  );

}
```

The Board component represents the game board, consisting of 9 squares.

handleClick Function:

Handles a click on a square. It first checks if the square is already filled or if there is already a winner. If so, it does nothing.

If the square is empty and the game is ongoing, it updates the square with the current player's symbol (X or O), and calls handleHistory to update the game history.

Winner Calculation:

The winner variable stores the result of the calculateWinner function. If there's a winner, it displays the winner; otherwise, it displays whose turn is next.

Rendering:

The board is rendered as three rows of Square components, each corresponding to an index in the squares array.

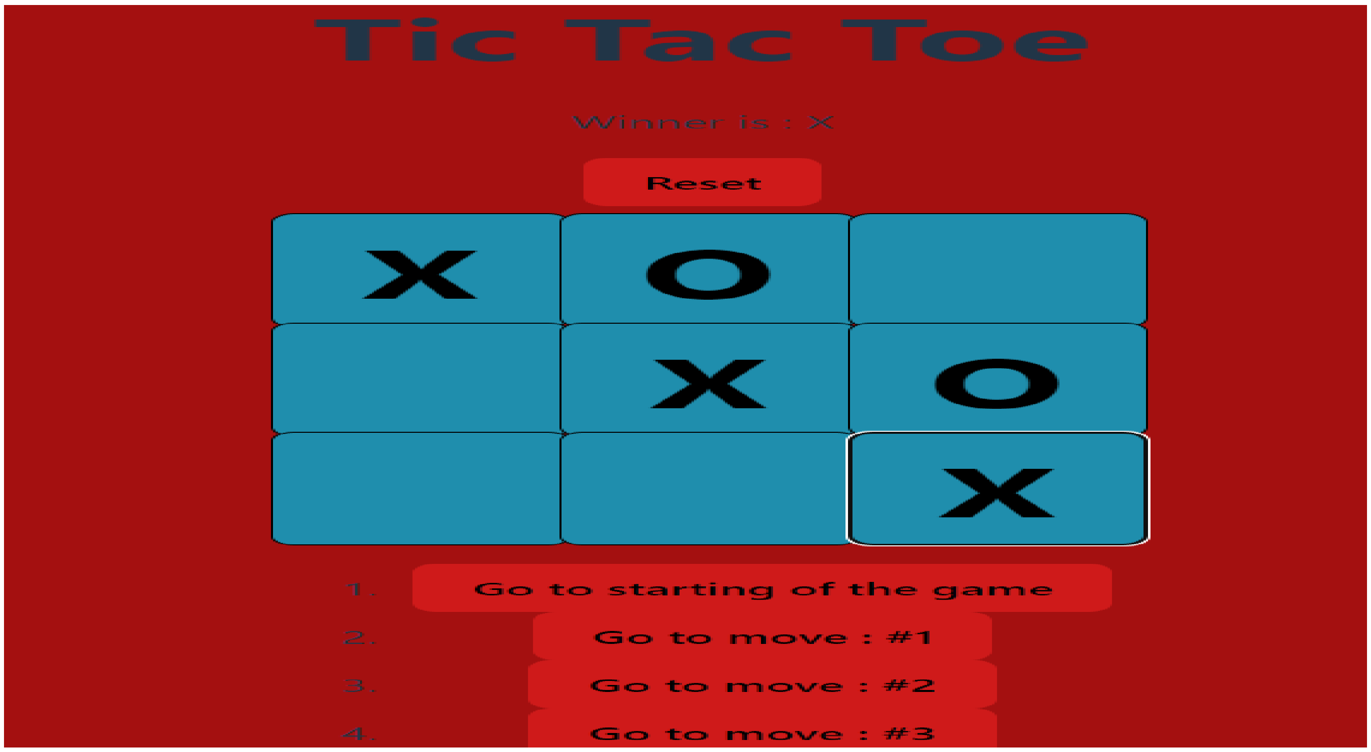
6. App Component:

```
function App() {

  let gameHeading = "Tic Tac Toe";

  return (

    <div

      style={{

        display: "flex",

        alignItems: "center",

        justifyContent: "center",
        flexDirection: "column",
      }}
    >
      <h1>{gameHeading}</h1>
      <TicTacToe />
    </div>
  );}
export default App;
```

The App component is the main entry point of the application. It displays the game heading and renders the TicTacToe component.

It uses inline styles to center the game on the screen.

Screenshot:



Final Project

Project Introduction-

This project is a blogging website built using the MERN stack (MongoDB, Express.js, React, and Node.js), providing a dynamic platform for users to create, read, and manage blog posts. The website offers a seamless user experience with a structured layout and intuitive navigation.

Key Features:

- **Home Page:** The home page features a navbar at the top, allowing users to easily navigate between the home, new post, settings, and user profile pages. It also displays a list of various articles fetched directly from the database, giving users access to the latest blog posts.
- **New Post:** Users can create new articles by navigating to the new post page, where they can compose and publish their content directly to the website.
- **Settings:** The settings page allows users to update their personal information and provides an option to log out from the site, ensuring their profile remains secure and up-to-date.
- **User Profile:** The user profile page displays the user's name. Although the page is not fully functional yet, it lays the foundation for further enhancements and personalization.

This project demonstrates a comprehensive understanding of full-stack development, showcasing the integration of front-end and back-end technologies to create a fully functional and interactive blogging platform.

TechStack used in the project – MongoDB, Node.js, Express.js, React

MongoDB : is a NoSQL database that stores data in a flexible, JSON-like format called BSON (Binary JSON). Unlike traditional relational databases, MongoDB does not require a predefined schema, making it highly adaptable to changes in data structure over time. This flexibility is especially advantageous for modern web applications where data needs are constantly evolving.

Key Advantages of MongoDB:

1. **Schema Flexibility:** MongoDB allows for dynamic schemas, meaning the structure of the data can change over time without requiring significant alterations to the database. This is ideal for projects that are in continuous development.
2. **Scalability:** MongoDB supports horizontal scaling, allowing it to handle large volumes of data by distributing the load across multiple servers. This makes it a popular choice for applications that require high availability and performance.
3. **Rich Query Language:** MongoDB offers a powerful query language that supports a wide range of operations, including filtering, sorting, and aggregation, enabling complex data retrieval and manipulation.
4. **Document-Oriented Storage:** Data in MongoDB is stored in documents, which are more aligned with how data is represented in modern applications. This makes the interaction between the application and database more natural and efficient.
5. **High Performance:** MongoDB is optimized for high-throughput read and write operations, making it suitable for applications with heavy traffic and large datasets.

In my MERN blogging website, MongoDB serves as the backbone for data storage. The database is structured to store:

- **Articles:** Each article is stored with details such as the title, author, body content, and timestamps for when the article was created and last updated.
- **Comments:** Comments associated with articles are stored with their body content and author information.
- **Users:** The user data, including the list of users, their email addresses, and securely hashed passwords (using the bcrypt library), is stored in MongoDB, ensuring both security and efficient data retrieval.

This implementation of MongoDB ensures that my blogging platform is both scalable and flexible, capable of handling varying data requirements as the project evolves.

Node.js is a JavaScript runtime built on Chrome's V8 engine that allows developers to execute JavaScript code server-side. It provides an event-driven, non-blocking I/O model, making it highly efficient and suitable for building scalable network applications. Node.js is designed to handle numerous simultaneous connections with high performance, which is ideal for real-time applications and services.

Key Features of Node.js:

- **Asynchronous and Event-Driven:** Uses non-blocking I/O operations to handle multiple requests efficiently.
- **Single Programming Language:** Enables server-side development using JavaScript, unifying client and server-side code.
- **Package Management:** Uses npm (Node Package Manager) to manage and distribute libraries and dependencies.

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It simplifies the development of web servers and APIs by offering a higher-level abstraction over Node.js's core HTTP module.

Key Features of Express.js:

- **Middleware:** Allows the use of middleware functions to process requests, modify requests and responses, and handle errors.
- **Routing:** Provides a straightforward API for defining routes and handling HTTP requests.
- **Configuration and Customization:** Easily configurable with various options for templating engines, static file serving, and more.

Usage in My Project: In my MERN blogging website, **Node.js** serves as the server-side runtime, handling various backend operations. **Express.js** is used to build the RESTful API, manage routing, and handle HTTP requests efficiently. Together, they enable seamless interaction between the front-end and the database, providing a responsive and scalable backend for the application.

Below is the logic of my code.

- **User Management**

  Base URL: All endpoints are relative to the base URL: /api

Authentication: Endpoints that require authentication use JWT (JSON Web Token) in the Authorization header.

Error Handling: Error responses follow standard HTTP status codes. Details of error responses are included in each endpoint's description.

Endpoints

1. **User Login**

URL: /users/login

Method: POST

Description: Authenticate a user to obtain a JWT token.

Request Body:

```
{

  "email": "user@example.com",

  "password": "password"

}
```

Success Response:

Code: 200 OK

Content:

```
{

   "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOiIxMjM0NTY3ODkwIiwiaWF0IjoxNTE2
MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c"

  }
```

- **Error Responses:**
  - **Code:** 401 Unauthorized
    - **Content:** { "error": "Invalid credentials" }
  - **Code:** 400 Bad Request
    - **Content:** { "error": "Missing email or password" }

2. User Registration

- **URL:** /users
- **Method:** POST
- **Description:** Register a new user.
- **Request Body:**

```
{
  "name": "John Doe",
  "email": "user@example.com",
  "password": "password"
}
```

- **Success Response:**
  - **Code:** 200 OK
  - **Content:**

```
{
  "message": "User registered successfully"
 }
```

- **Error Responses:**
  - **Code:** 400 Bad Request
    - **Content:** { "error": "Email already exists" }
  - **Code:** 422 Unprocessable Entity
    - **Content:** { "error": "Invalid input data" }

3. Get Current User

- **URL:** /user
- **Method:** GET
- **Description:** Retrieve information about the currently authenticated user.
- **Headers:**
  - Authorization: Bearer <token>
- **Success Response:**
  - **Code:** 200 OK
  - **Content:**

```
{
  "user": {
    "id": "1234567890",
    "name": "John Doe",
    "email": "user@example.com"
  }
}
```

- **Error Responses:**
  - **Code:** 401 Unauthorized
    - **Content:** { "error": "Unauthorized: Invalid token" }

4. Update User

- **URL:** /user
- **Method:** PUT
- **Description:** Update the information of the currently authenticated user.
- **Headers:**
  - Authorization: Bearer <token>
- **Request Body:**

```
{
  "name": "Updated Name"
}
```

- **Success Response:**
  - **Code:** 200 OK
  - **Content:**

```
{
  "message": "User updated successfully"

 }
```

- **Error Responses:**
  - **Code:** 401 Unauthorized
    - **Content:** { "error": "Unauthorized: Invalid token" }
  - **Code:** 400 Bad Request
    - **Content:** { "error": "Invalid input data" }

All endpoints are relative to the base URL: /api/articles

Authentication

Endpoints

1. Create Article

- **URL:** /
- **Method:** POST
- **Description:** Create a new article.
- **Authentication:** Requires a valid JWT in the Authorization header.
- **Request Body:**

```
{

  "article": {

    "title": "Title of the Article",

    "description": "Description of the article",

    "body": "Body content of the article",

    "tagList": ["tag1", "tag2"]

  }

 }
```

- **Success Response:**

- **Code:** 200 OK
- **Content:**

```
{
  "article": {
    "title": "Title of the Article",
    "description": "Description of the article",
    "body": "Body content of the article",
    "author": {
      "id": "user_id",
      "username": "username"
    },
    "tagList": ["tag1", "tag2"],
    "createdAt": "2024-07-24T12:00:00Z",
    "updatedAt": "2024-07-24T12:00:00Z"
  }
 }
```

- **Error Responses:**
  - **Code:** 400 Bad Request
    - **Content:** { "message": "All fields are required" }
  - **Code:** 401 Unauthorized
    - **Content:** { "error": "Unauthorized: Invalid token" }

2. Feed Articles

- **URL:** /
- **Method:** GET

- **Description:** Fetch all articles.
- **Success Response:**
  - **Code:** 200 OK
  - **Content:** Array of articles in the following format:

```
[
  {
   "title": "Title of the Article",
   "description": "Description of the article",
   "body": "Body content of the article",
   "author": {
     "id": "user_id",
     "username": "username"
   },
   "tagList": ["tag1", "tag2"],
   "createdAt": "2024-07-24T12:00:00Z",
   "updatedAt": "2024-07-24T12:00:00Z"
  },
  // More articles...
 ]
```

- **Error Responses:**
  - **Code:** 500 Internal Server Error
    - **Content:** { "error": "Error fetching articles" }

3. Get Article by Slug

- **URL:** /:slug

- **Method:** GET
- **Description:** Fetch a single article by its slug.
- **Parameters:**
    - slug: Unique slug of the article.
- **Success Response:**
    - **Code:** 200 OK
    - **Content:**

```
{

  "article": {

    "title": "Title of the Article",

    "description": "Description of the article",

    "body": "Body content of the article",

    "author": {

      "id": "user_id",

      "username": "username"

    },

    "tagList": ["tag1", "tag2"],

    "createdAt": "2024-07-24T12:00:00Z",

    "updatedAt": "2024-07-24T12:00:00Z"

  }

}
```

- **Error Responses:**
    - **Code:** 401 Unauthorized
        - **Content:** { "message": "Article Not Found" }

JWT Verification Middleware

The verifyJWT middleware function is used to authenticate and authorize requests by verifying a JSON Web Token (JWT) provided in the Authorization header. It decodes the JWT using a secret key (ACCESS_TOKEN_SECRET) and attaches the decoded user information to the request object (req) for further processing by subsequent middleware or route handlers.

Functionality

The middleware performs the following tasks:

1. **Extracts Authorization Header:** Retrieves the JWT from the Authorization header of the HTTP request.
2. **Validates JWT Format:** Checks if the JWT starts with the string "Token ".
3. **Verifies JWT:** Uses jwt.verify method from the jsonwebtoken package to verify the validity and authenticity of the JWT against the ACCESS_TOKEN_SECRET.
4. **Handles Authentication Errors:** Returns appropriate HTTP responses (401 Unauthorized or 403 Forbidden) if the JWT is missing, malformed, expired, or invalid.
5. **Attaches User Information:** If the JWT is valid, extracts the user ID, email, and hashed password from the decoded JWT payload and attaches them to the req object (req.userId, req.userEmail, req.userHashedPass).
6. **Calls Next Middleware:** Calls the next() function to pass control to the next middleware or route handler in the request processing pipeline.

Usage

To use verifyJWT middleware in your Express routes:

1. **Import the Middleware:**

```
const verifyJWT = require('../middleware/verifyJWT');
```

2. **Apply the Middleware to Routes:** Add verifyJWT as middleware to any route or group of routes that require authentication and authorization:

```
router.get('/protected-route', verifyJWT, (req, res) => {

  // Route handler logic for authenticated requests

});
```

Error Handling

- **401 Unauthorized:** Returned if the JWT is missing or does not start with "Token ".
- **403 Forbidden:** Returned if the JWT is invalid or expired.

Example

```
// Example route using verifyJWT middleware

const express = require('express');

const router = express.Router();

const verifyJWT = require('../middleware/verifyJWT');



// Protected route example

router.get('/protected-route', verifyJWT, (req, res) => {

    // Only reaches here if JWT is valid and user is authenticated

    res.json({ message: 'Authenticated user' });

});



module.exports = router;
```

Security Considerations

- **Token Security:** Ensure that ACCESS_TOKEN_SECRET is securely stored and not exposed in your codebase.
- **JWT Expiry:** Consider implementing token expiration and refresh mechanisms for improved security.

Dependencies Used

Production Dependencies

1. bcrypt (^5.1.1)

- **Description:** Library for hashing passwords using bcrypt algorithm.
- **Link:** npm bcrypt

2. cors (^2.8.5)

- **Description:** Middleware for enabling Cross-Origin Resource Sharing (CORS) in Express.js.
- **Link:** npm cors

3. dotenv (^16.4.5)

- **Description:** Module for loading environment variables from a .env file into process.env.
- **Link:** npm dotenv

4. express (^4.19.2)

- **Description:** Fast, unopinionated, minimalist web framework for Node.js.
- **Link:** npm express

5. jsonwebtoken (^9.0.2)

- **Description:** JSON Web Token (JWT) implementation for Node.js.
- **Link:** npm jsonwebtoken

6. mongodb (^6.8.0)

- **Description:** Official MongoDB driver for Node.js.
- **Link:** npm mongodb

7. mongoose (^8.5.0)

- **Description:** Elegant MongoDB object modeling for Node.js.
- **Link:** npm mongoose

8. mongoose-unique-validator (^5.0.1)

- **Description:** Plugin for Mongoose that adds pre-save validation for unique fields.
- **Link:** npm mongoose-unique-validator

9. slugify (^1.6.6)

- **Description:** Library to create slugs from strings with options for customization.
- **Link:** npm slugify

Development Dependencies

1. nodemon (^3.1.4)

- **Description:** Utility that monitors changes in your Node.js application and automatically restarts the server.
- **Link:** npm nodemon

Article Schema

Overview

The articleSchema defines the structure of MongoDB documents for articles using Mongoose, providing a blueprint for storing and querying articles in your application.

Fields

1. slug

- **Type:** String
- **Description:** Unique slug generated from the title using slugify.
- **Attributes:**
  - unique: Ensures uniqueness across documents.
  - lowercase: Converts the slug to lowercase.
  - index: Creates an index for efficient querying.

2. title

- **Type:** String
- **Description:** Title of the article.
- **Attributes:**
  - required: Field must be provided for document creation.

3. description

- **Type:** String
- **Description:** Short description or summary of the article.
- **Attributes:**
  - required: Field must be provided for document creation.

4. body

- **Type:** String
- **Description:** Main content or body of the article.
- **Attributes:**
  - required: Field must be provided for document creation.

5. tagList

- **Type:** Array of Strings
- **Description:** List of tags associated with the article.

6. author

- **Type:** ObjectId
- **Description:** Reference to the User who authored the article.
- **Attributes:**

    - ref: Refers to the User model in Mongoose.

7. favoritesCount

- **Type:** Number
- **Description:** Number of times the article has been favorited.
- **Default:** 0

8. comments

- **Type:** Array of ObjectIds
- **Description:** References to comments associated with the article.
- **Attributes:**
    - ref: Refers to the Comment model in Mongoose.

Plugins

1. mongoose-unique-validator

- **Description:** Plugin for Mongoose that adds pre-save validation for unique fields, used here for slug field.

Middleware

pre('save')

- **Description:** Middleware that runs before saving an article document.
- **Functionality:** Generates a unique slug based on the title using slugify.

Methods

toArticleResponse(user)

- **Description:** Method to format article data into a response object.
- **Parameters:**
    - user: Logged-in user object for determining favorite status.
- **Returns:** Formatted article data including author details formatted via toProfileJSON.

addComment(commentId)

- **Description:** Method to add a comment reference to the article's comments array.
- **Parameters:**
    - commentId: ObjectId of the comment to add.
- **Returns:** Promise resolving to the saved article document.

removeComment(commentId)

- **Description:** Method to remove a comment reference from the article's comments array.
- **Parameters:**

    - commentId: ObjectId of the comment to remove.
- **Returns:** Promise resolving to the saved article document.

User Schema

Overview

The userSchema defines the structure of MongoDB documents for users using Mongoose, providing a blueprint for storing and querying user information in your application.

Fields

1. username

- **Type:** String
- **Description:** User's username.
- **Attributes:**
    - required: Field must be provided for document creation.
    - unique: Username must be unique across documents.
    - lowercase: Converts the username to lowercase.

2. password

- **Type:** String
- **Description:** User's password.
- **Attributes:**
    - required: Field must be provided for document creation.

3. email

- **Type:** String
- **Description:** User's email address.
- **Attributes:**
    - required: Field must be provided for document creation.
    - unique: Email must be unique across documents.
    - lowercase: Converts the email address to lowercase.
    - match: Validates that the email format is correct (/\\S+@\\S+\\.\\S+/).
    - index: Optimizes query performance by creating an index.

Methods

generateAccessToken()

- **Description:** Method to generate a JWT access token for authentication.
- **Returns:** Access token signed with ACCESS_TOKEN_SECRET and containing user ID, email, and password (payload).
- **Expiration:** Token expires in 1 day ("1d").

toUserResponse()

- **Description:** Method to format user data into a response object.
- **Returns:** Object containing username, email, and JWT access token (token) generated using generateAccessToken().

toProfileJSON(user)

- **Description:** Method to format user data for a user profile response.
- **Parameters:**
    - user: Logged-in user object (not used in the current implementation).
- **Returns:** Object containing username, bio (if available), image (if available), and following count.

Comment Schema

Overview

The commentSchema defines the structure of MongoDB documents for comments using Mongoose, providing a blueprint for storing and querying comments in relation to articles and users in your application.

Fields

1. body

- **Type:** String
- **Description:** Content or text of the comment.
- **Attributes:**
    - required: Field must be provided for document creation.

2. author

- **Type:** ObjectId
- **Description:** Reference to the User who authored the comment.
- **Attributes:**
    - ref: Refers to the User model in Mongoose.

3. article

- **Type:** ObjectId
- **Description:** Reference to the Article to which the comment belongs.
- **Attributes:**
    - ref: Refers to the Article model in Mongoose.

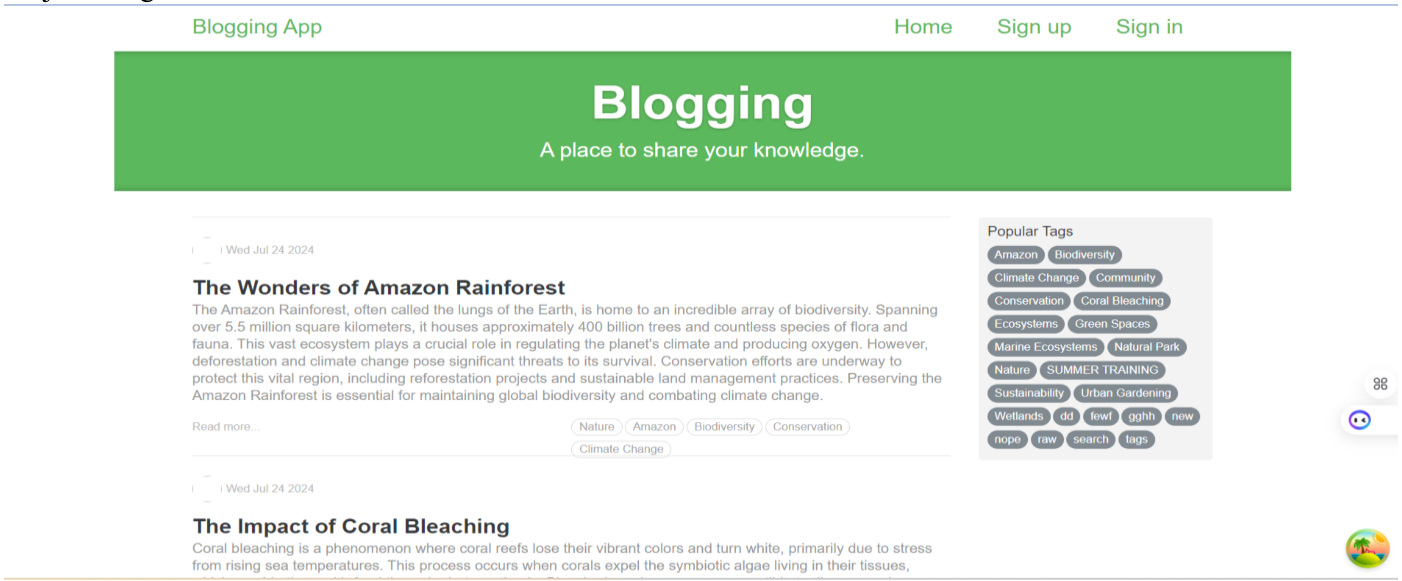
Methods

toCommentResponse(user)

- **Description:** Method to format comment data into a response object.
- **Parameters:**
  - user: Logged-in user object for determining author details.
- **Returns:** Object containing comment ID, body, timestamps (createdAt, updatedAt), and formatted author details using toProfileJSON() method of the User model.

GitHub Link - https://github.com/SANVIOJHA/blogging

Project Images –



Blogging App Home Sign up Sign in

Blogging

A place to share your knowledge.

Wed Jul 24 2024

The Wonders of Amazon Rainforest

The Amazon Rainforest, often called the lungs of the Earth, is home to an incredible array of biodiversity. Spanning over 5.5 million square kilometers, it houses approximately 400 billion trees and countless species of flora and fauna. This vast ecosystem plays a crucial role in regulating the planet's climate and producing oxygen. However, deforestation and climate change pose significant threats to its survival. Conservation efforts are underway to protect this vital region, including reforestation projects and sustainable land management practices. Preserving the Amazon Rainforest is essential for maintaining global biodiversity and combating climate change.

Read more... Nature Amazon Biodiversity Conservation Climate Change

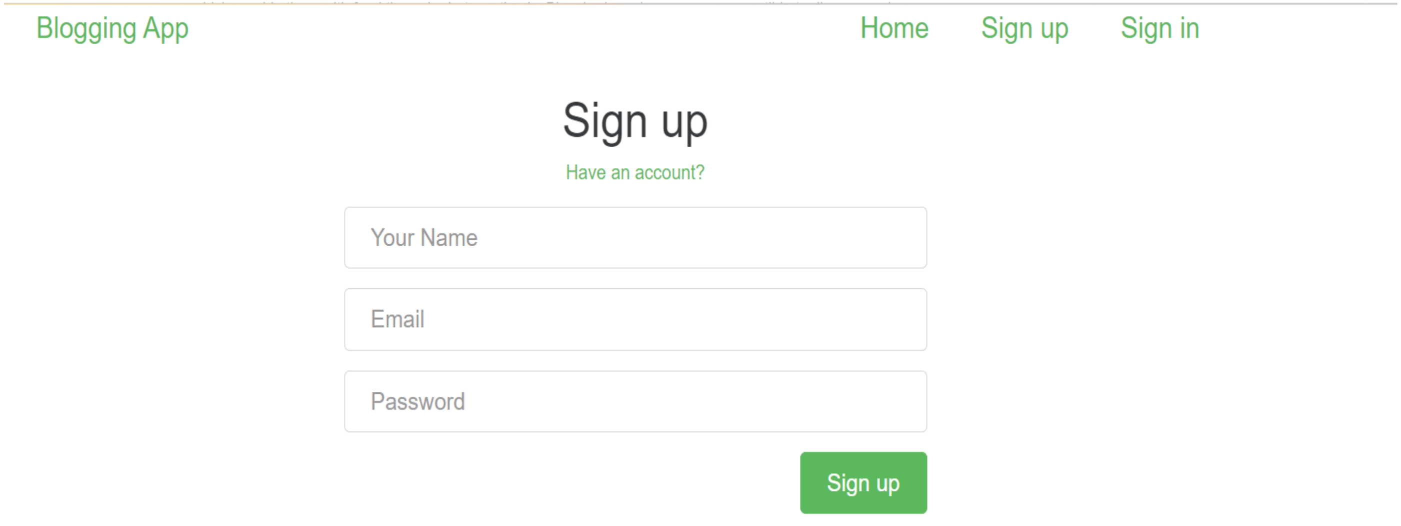
Wed Jul 24 2024

The Impact of Coral Bleaching

Coral bleaching is a phenomenon where coral reefs lose their vibrant colors and turn white, primarily due to stress from rising sea temperatures. This process occurs when corals expel the symbiotic algae living in their tissues,

Popular Tags

Amazon Biodiversity Climate Change Community Conservation Coral Bleaching Ecosystems Green Spaces Marine Ecosystems Natural Park Nature SUMMER TRAINING Sustainability Urban Gardening Wetlands dd fewf gghh new nope raw search tags



Blogging App Home Sign up Sign in

Sign up

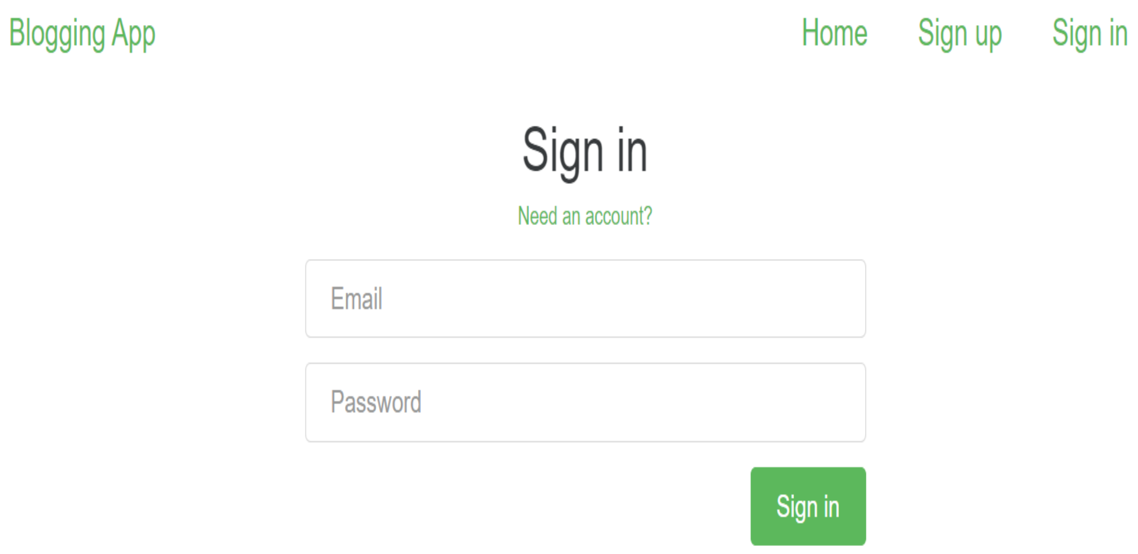
Have an account?

Your Name

Email

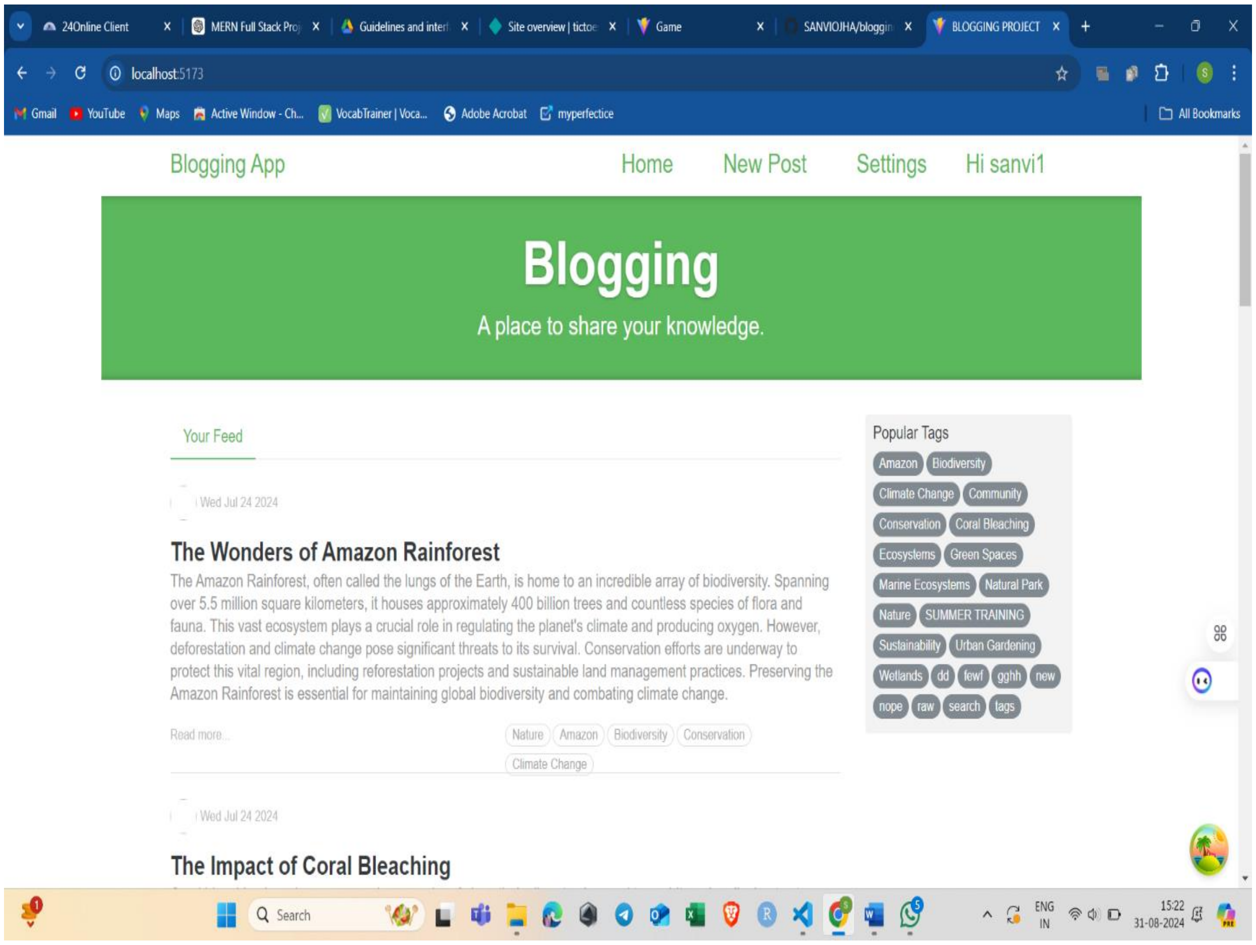
Password

Sign up



Sign in

Need an account?



Blogging

A place to share your knowledge.

Your Feed

Wed Jul 24 2024

The Wonders of Amazon Rainforest

The Amazon Rainforest, often called the lungs of the Earth, is home to an incredible array of biodiversity. Spanning over 5.5 million square kilometers, it houses approximately 400 billion trees and countless species of flora and fauna. This vast ecosystem plays a crucial role in regulating the planet's climate and producing oxygen. However, deforestation and climate change pose significant threats to its survival. Conservation efforts are underway to protect this vital region, including reforestation projects and sustainable land management practices. Preserving the Amazon Rainforest is essential for maintaining global biodiversity and combating climate change.

Read more...

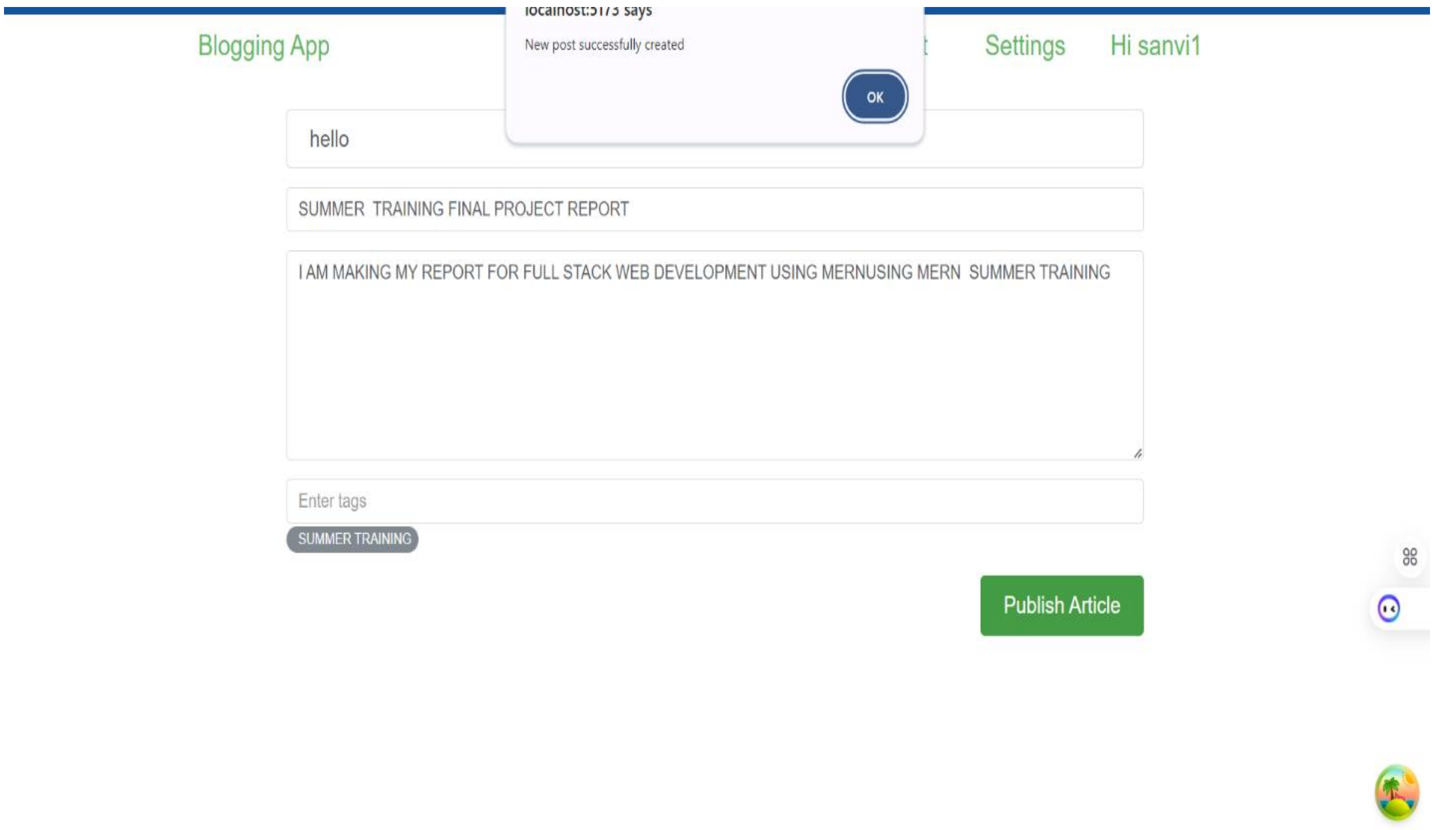
Nature Amazon Biodiversity Conservation Climate Change

Wed Jul 24 2024

The Impact of Coral Bleaching

Popular Tags

Amazon Biodiversity Climate Change Community Conservation Coral Bleaching Ecosystems Green Spaces Marine Ecosystems Natural Park Nature SUMMER TRAINING Sustainability Urban Gardening Wetlands dd fewf gghh new nope raw search tags


Blogging App

localhost:5173 says

New post successfully created

OK

Settings Hi sanvi1

hello

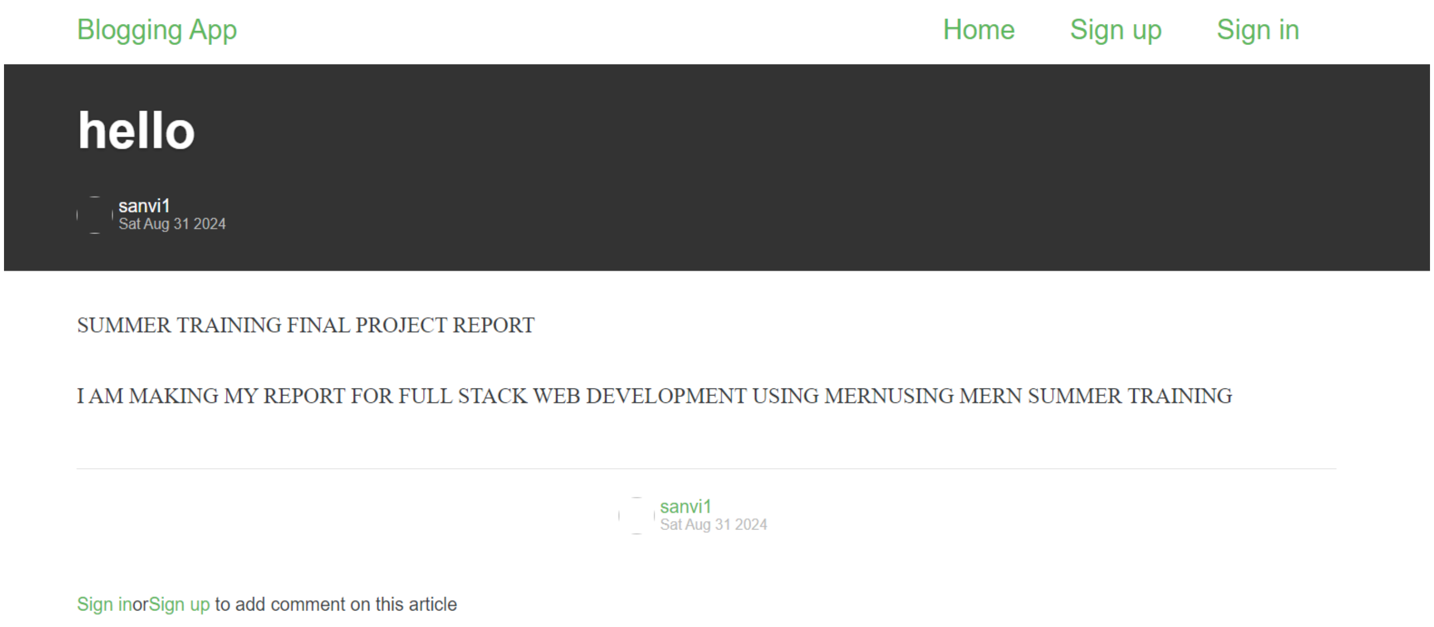
SUMMER TRAINING FINAL PROJECT REPORT

I AM MAKING MY REPORT FOR FULL STACK WEB DEVELOPMENT USING MERNUSING MERN SUMMER TRAINING

Enter tags

SUMMER TRAINING

Publish Article


Blogging App Home Sign up Sign in

hello

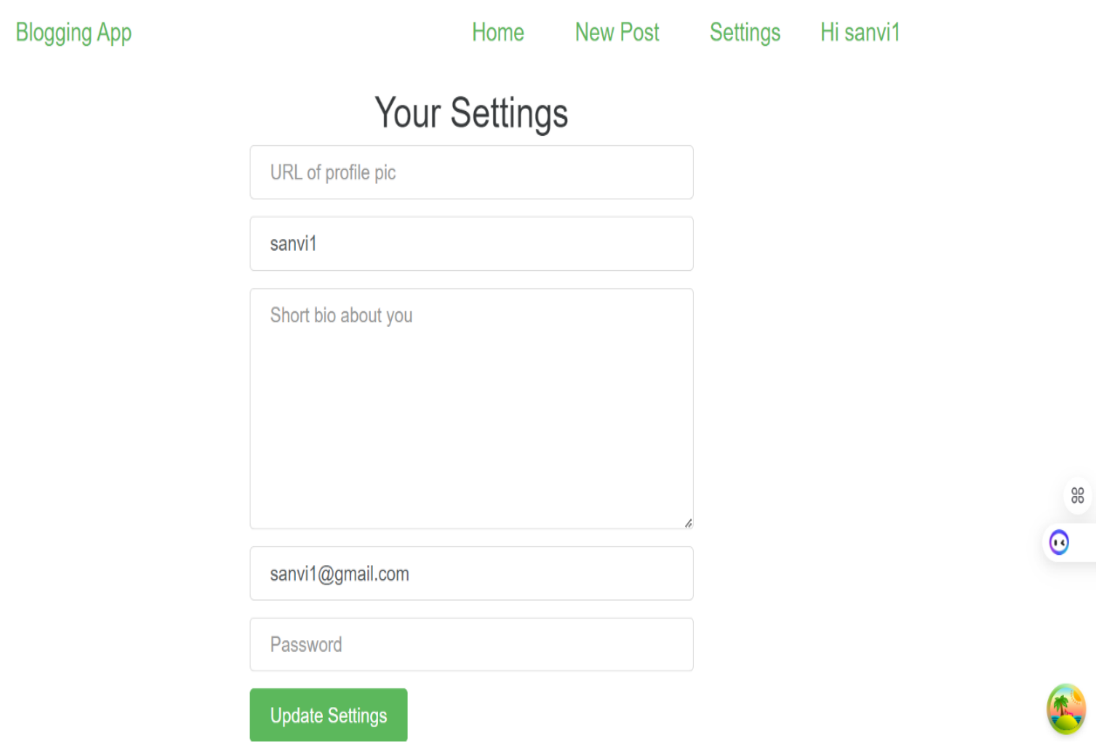
sanvi1
Sat Aug 31 2024

SUMMER TRAINING FINAL PROJECT REPORT

I AM MAKING MY REPORT FOR FULL STACK WEB DEVELOPMENT USING MERNUSING MERN SUMMER TRAINING

sanvi1
Sat Aug 31 2024

Sign inorSign up to add comment on this article


Blogging App Home New Post Settings Hi sanvi1

Your Settings

URL of profile pic

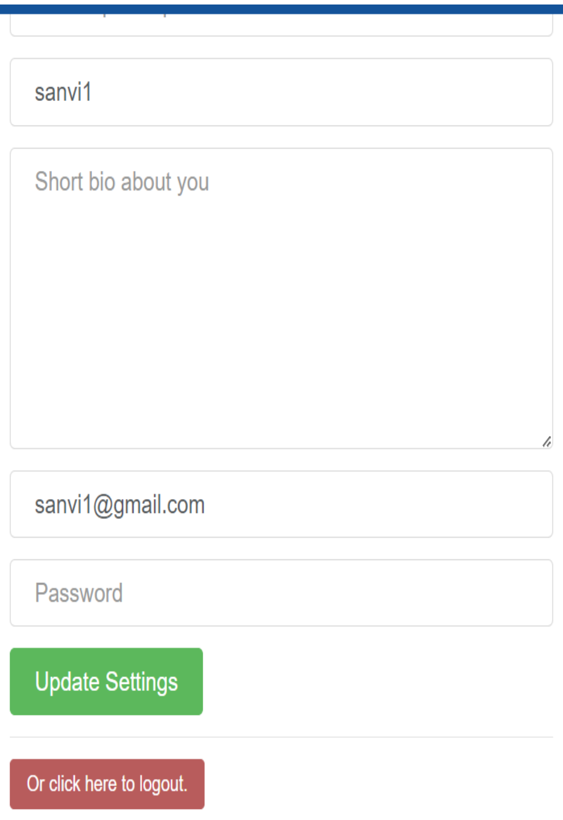
sanvi1

Short bio about you

sanvi1@gmail.com

Password

Update Settings


sanvi1

Short bio about you

sanvi1@gmail.com

Password

Update Settings

Or click here to logout.

CONCLUSION

The summer training in Full Stack Development using the MERN stack has been an extremely valuable learning experience, greatly enhancing my technical abilities and deepening my knowledge of web development. This project pushed me to step outside my comfort zone, as I transitioned from a data science background to mastering the intricacies of full stack development. By engaging in hands-on projects, I developed proficiency in both frontend and backend technologies, as well as database management and deployment processes. The mini projects, along with the major Blogging website project, provided practical experience in addressing real-world challenges. These tasks helped me sharpen my problem-solving skills and reinforced the importance of designing user-friendly interfaces and managing data efficiently. The skills and knowledge I have gained through this training will be crucial for my future academic and professional pursuits, particularly in my upcoming placement projects.

In summary, this training has not only strengthened my technical foundation but also built my confidence in handling complex development tasks. I am now better equipped to meet the evolving demands of the tech industry.

REFERENCES

Sahil Chopra sir’s: https://obvious-cosmos-bed.notion.site/API-Documentation-a346a55b65af41d69d3ab229dbcd0173

GitHub: https://github.com/sash9696/cipher-mern-stack

Chrome: https://www.oracle.com/in/database/mern-stack/#what