**A Web Technology Mini Project Report**

**on**

“CHAT APPLICATION”

Submitted to the

Savitribai Phule Pune University

In partial fulfillment of

“Artificial Intelligence and Machine Learning”

By  
  
**Rucha Salunke [33561]**

Under the guidance of

**Subject In charge: Prof.Gauri.Mathad**



Department Of Artificial Intelligence and Machine Learning

PES’s Modern College of Engineering, Shivaji Nagar, Pune-411005, Maharashtra, India 2024-2025



CERTIFICATE

This is to certify that the Web Technology Mini Project report entitled **“ Chat Application ”** being submitted by **Rucha Salunke** is a record of bonafide work carried out by him/her under the supervision and guidance of **Prof.Gauri Mathad Ma’am** in partial fulfillment of the requirement for **TE (**Artificial Intelligence and Machine Learning**) – 2019 course** of Savitribai Phule Pune University, Pune in the academic year 2024-2025

Date:22/10/24

Place: Pune

Prof.Mrs.Gauri.Mathad Guide of Web Technology

Dr.Mrs.S.D.Deshpande

Head of the Department

# ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to everyone who contributed to the successful completion of this project management system.

First and foremost, I extend my sincere thanks to my project supervisor **Prof. Gauri Ma’am**, for their invaluable guidance and support throughout the development process. Their insights and feedback were instrumental in shaping the project and overcoming challenges.

I am also deeply grateful to my teammates, for their collaboration, dedication, and hard work. Each member's unique contributions enriched this project and made the journey truly enjoyable.

A special thank you goes to college Professor for providing assistance and resources that facilitated my work. Your expertise and advice were crucial in navigating the complexities of this project.

Lastly, I would like to acknowledge my family and friends for their unwavering support and encouragement. Their understanding and patience helped me stay focused and motivated.

Thank you all for your contributions and support.

Rucha Salunke [33561]

**INTRODUCTION**

This project is a **real-time chat application** developed using the **MERN stack**, which includes **MongoDB**, **Express.js**, **React.js**, and **Node.js**. The primary goal of the project is to create a platform that allows users to communicate in real-time through web-based chat rooms. Users can send and receive messages instantly and view chat histories. The application showcases the integration of modern web technologies to provide a seamless user experience, emphasizing responsiveness and real-time interactivity.

**Project Features**

This chat application incorporates the following key features:

1. User Authentication: Users can register and log in to the application. Authentication is handled securely, allowing access to chat rooms only for registered users.
2. Real-Time Communication: Messages between users are transmitted in real time using Socket.io. This ensures that users can send and receive messages instantly, making the chat experience highly responsive.
3. Chat Rooms: Users can join different chat rooms, enabling group chats or topic-specific discussions.
4. Message History: The application stores chat history in MongoDB, allowing users to access previous conversations even after they log out.

**ABSTRACT**

This project presents a real-time chat application developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js). The application facilitates real-time communication between users by allowing them to join chat rooms, send instant messages, and view message histories. Key technologies such as Socket.io are integrated to enable real-time, bidirectional communication between the server and clients, ensuring that messages are transmitted with minimal delay.

The back-end of the application is powered by Node.js and Express.js, where user authentication, chat room management, and message handling are implemented. MongoDB serves as the database, storing user data, chat room details, and message histories, providing a flexible and scalable solution for managing unstructured data. The front-end is built using React.js, which offers a dynamic and responsive interface, enabling seamless interaction between users in real-time.

The project demonstrates several essential features, including secure user authentication, real-time messaging, and persistent chat histories. The application is designed with scalability and responsiveness in mind, offering a practical solution for modern web-based communication systems. Through the use of the MERN stack, this project highlights the integration of front-end and back-end technologies to create a fully functional, interactive web application.

**OBJECTIVE**

* **Develop a Real-Time Chat Application**: Build a fully functional chat platform that allows users to communicate instantly with each other using real-time messaging.
* **Implement the MERN Stack**: Utilize MongoDB, Express.js, React.js, and Node.js to create a full-stack web application, integrating both back-end and front-end functionalities efficiently.
* **Enable Real-Time Communication**: Integrate **Socket.io** to ensure seamless, bi-directional communication between users, allowing for the transmission of messages with minimal latency.
* **Provide Secure User Authentication**: Implement user registration and login features, ensuring that user data is protected and that only authenticated users can access the chat system.
* **Create Multiple Chat Rooms**: Allow users to join different chat rooms, enabling both group conversations and topic-specific chats.
* **Persist Chat Histories**: Store chat histories in MongoDB so users can access past messages when they re-enter the chat rooms, ensuring data persistence across sessions.
* **Build a Responsive User Interface**: Use React.js to develop an intuitive, responsive interface that provides users with a smooth and interactive chat experience across devices.
* **Ensure Scalability and Maintainability**: Design the application to be scalable, enabling the potential for future growth in terms of the number of users and chat rooms, and ensure that the codebase is maintainable for future updates or enhancements.
* **Optimize Performance and Usability**: Focus on minimizing message transmission delays and optimizing the overall performance of the application to provide a fast and user-friendly experience.

**BASIC STRUCTURE OF CHAT APPLICATION**

The project is organized into different folders and files that serve specific purposes. Here's an overview of the structure:

**1. Images Folder (images)**

This folder stores static image files used in the project, like snappy\_login.png and snappy.png. These images could be used for UI components, such as a login page background or logo.

**2. Public Folder (public)**

Purpose: Contains assets that will be served to the client by the React front-end.

Subfolders/Files:

node\_modules: Stores third-party libraries and packages used in the front-end.

src: Holds the React source code (though this isn't fully expanded in the image, it typically contains components, styles, etc.).

.env: Stores environment variables, typically configuration settings such as API keys or database URLs.

Dockerfile: Instructions for building a Docker container for the front-end application.

package.json: Defines project metadata, dependencies, and scripts for managing the front-end.

yarn.lock / package-lock.json: Lock files ensure consistent package versions are installed, maintaining stability across environments.

**3. Server Folder (server)**

Purpose: Contains the back-end logic for the application (Node.js/Express).

Subfolders/Files:

controllers: Contains files that handle the logic for API endpoints (e.g., creating, reading, updating, and deleting data).

models: Likely contains the data models or schemas for the application (possibly using a tool like Mongoose if MongoDB is in use).

routes: Defines the API routes, mapping HTTP requests to the corresponding controllers.

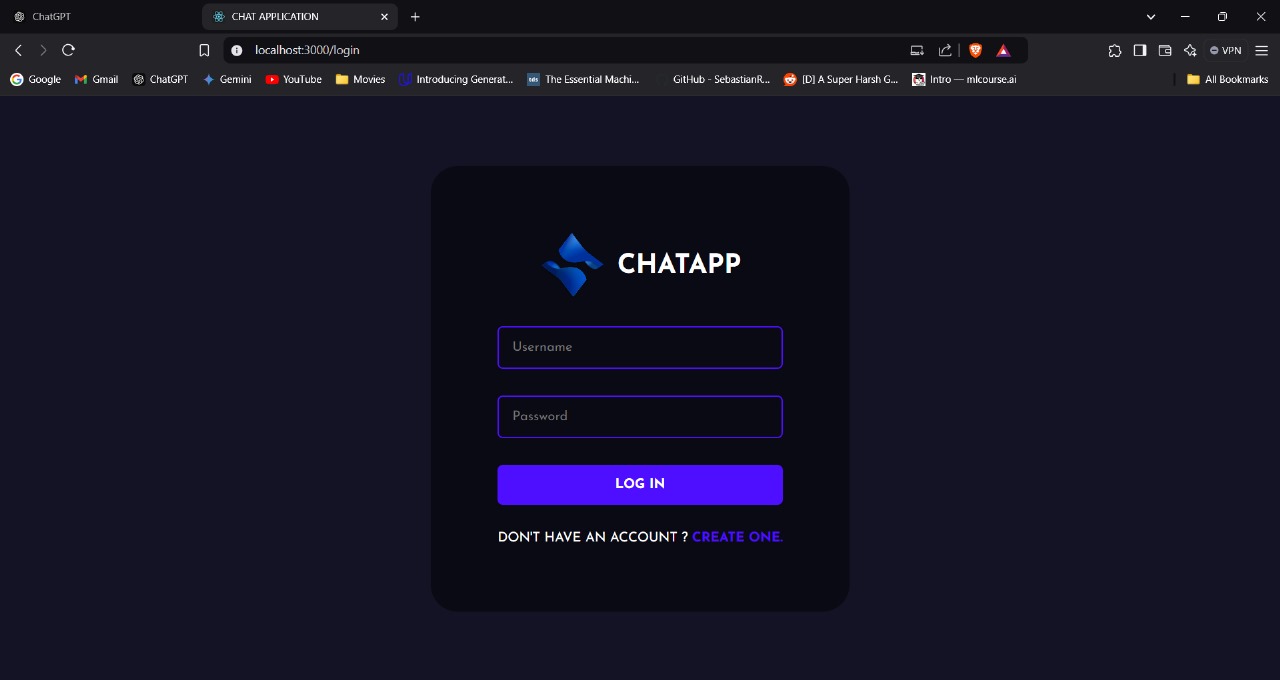
Dockerfile: Specifies how the back-end should be containerized using Docker.

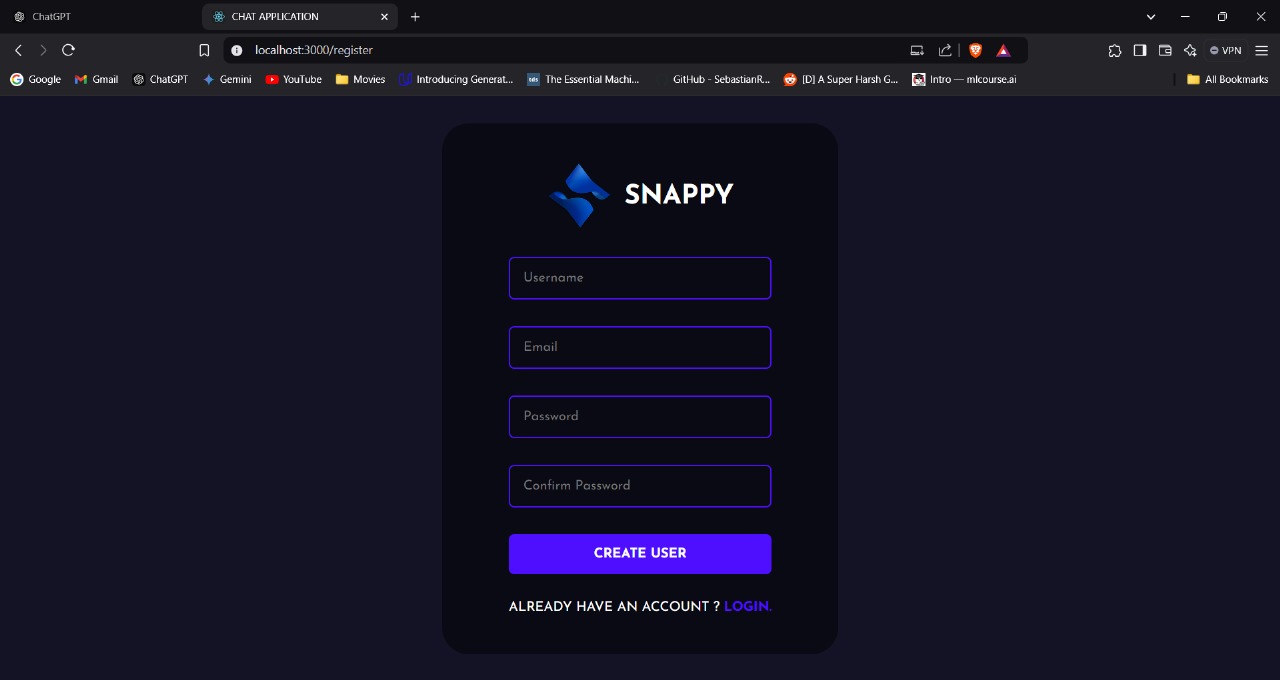
.env: Environment variables for back-end configurations like database credentials, server port, or authentication tokens.

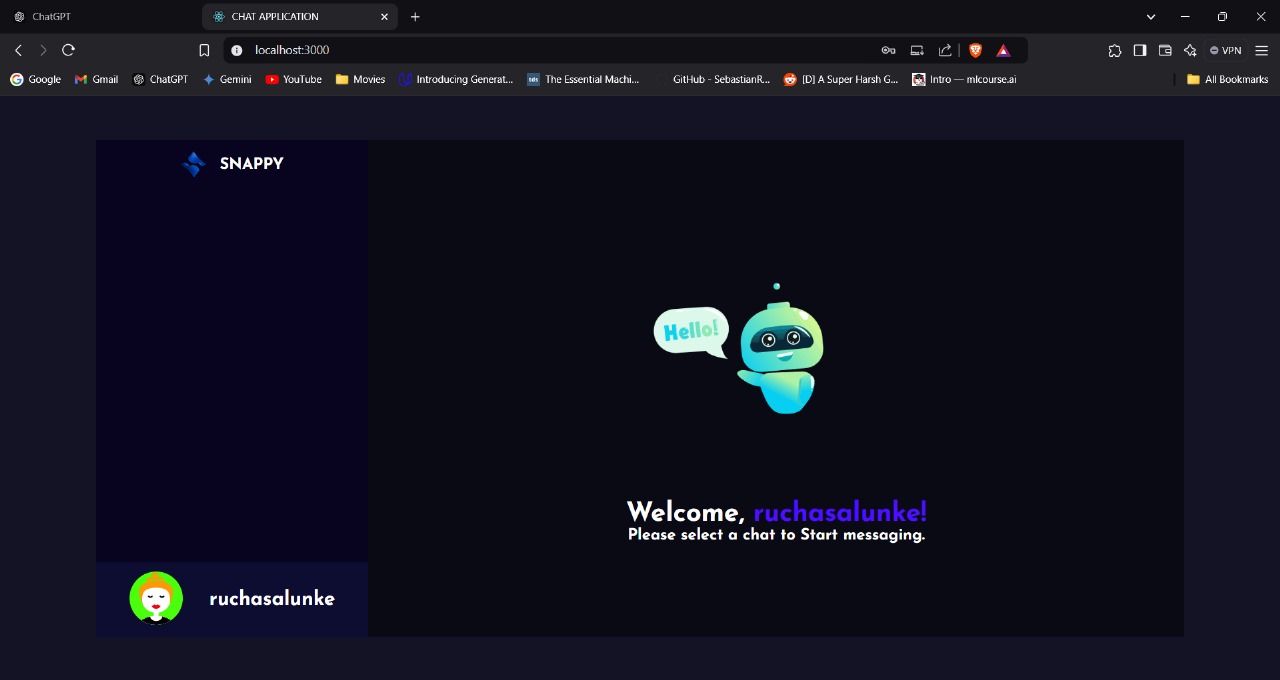
package.json / yarn.lock / package-lock.json: Configuration and dependency management files for the back-end (same function as in the public directory).

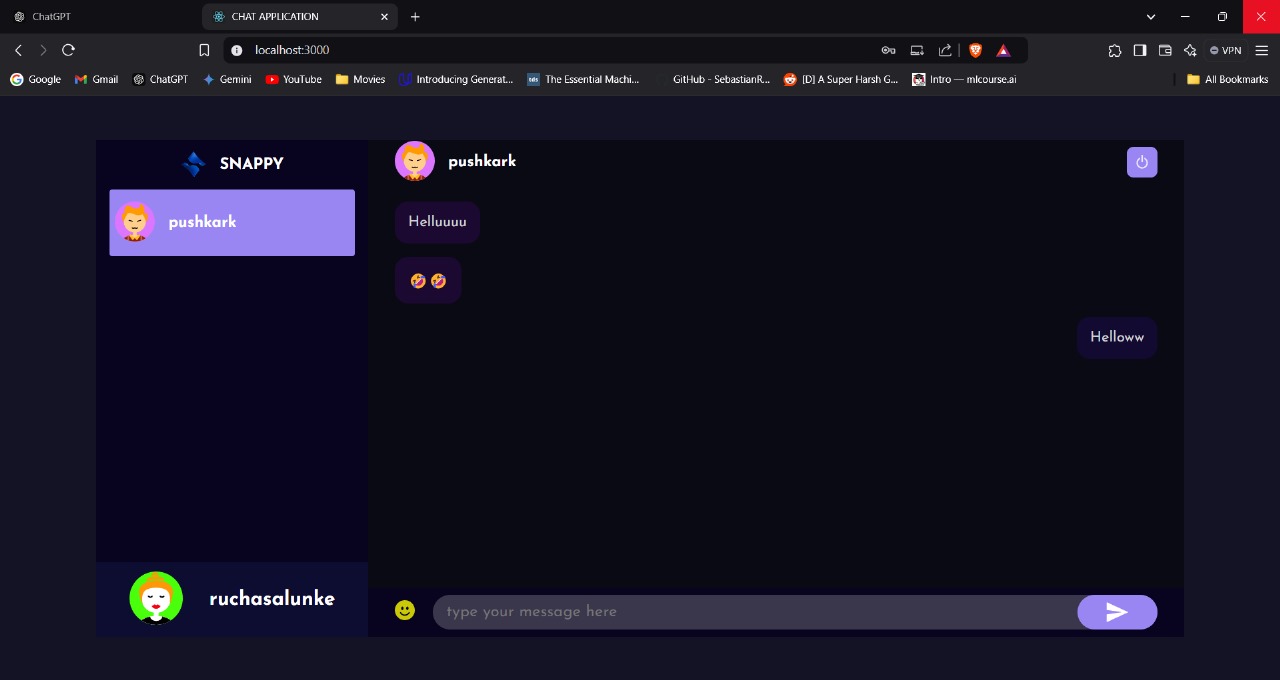
index.js: Entry point for the back-end application where the server setup (e.g., Express) and middleware are defined.

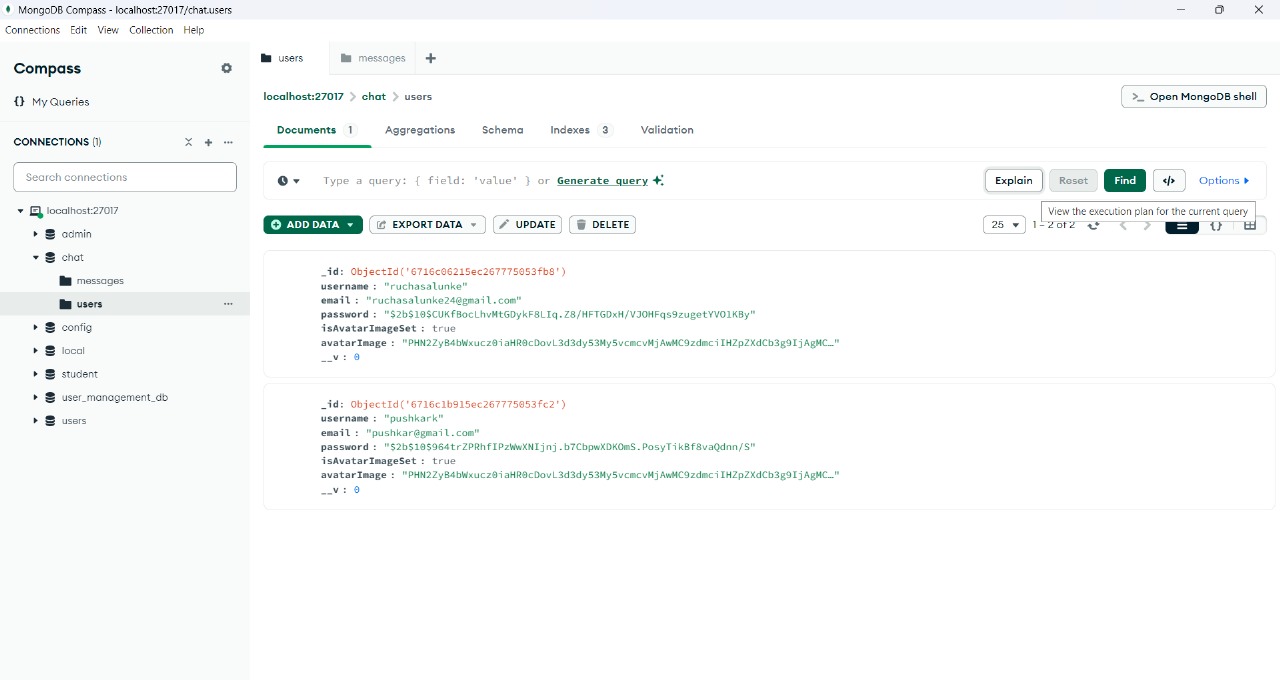
**OUTPUT**

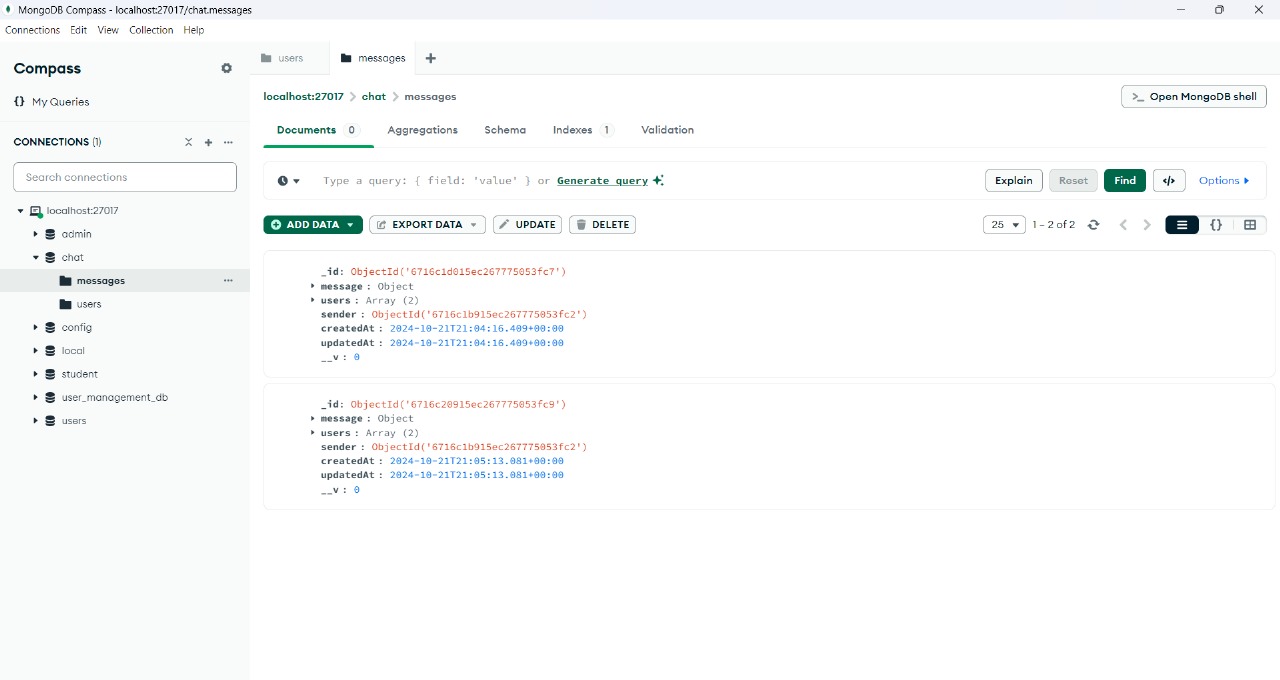
LOGIN PAGE

REGISTER PAGE

USER HOME PAGE

MESSAGES SENT

MONGODB USERS DATABASE

MONGODB MESSAGES DATABASE

**CONCLUSION**

In conclusion, this project successfully demonstrates the development of a **real-time chat application** using the **MERN stack** (MongoDB, Express.js, React.js, and Node.js). The application meets its objectives by providing a fully functional chat platform that allows users to communicate instantly in real-time, join multiple chat rooms, and view persistent message histories. Through the integration of **Socket.io**, real-time messaging is achieved, ensuring smooth, bi-directional communication between users with minimal latency.

The project emphasizes the practical use of modern web technologies, showcasing how front-end and back-end systems can be effectively integrated to create a responsive and user-friendly interface. The use of **MongoDB** ensures scalable and flexible data storage, while **React.js** provides an interactive user experience. **Node.js** and **Express.js** ensure efficient server-side operations and API handling, making the application both fast and reliable.

This chat application serves as a robust and scalable communication platform, laying the groundwork for future enhancements such as adding more advanced features like direct messaging, file sharing, or notifications. Overall, the project highlights the power of the MERN stack in building modern, real-time web applications and provides a solid foundation for future development and scalability.

**REFERENCES**

* **MongoDB Documentation**. (n.d.). Retrieved from <https://www.mongodb.com/docs/>
* Provides comprehensive details on MongoDB’s NoSQL database structure, setup, and usage for scalable applications.
* **Express.js Guide**. (n.d.). Retrieved from https://expressjs.com/en/guide/
* A guide to using Express.js for building web applications, including routing, middleware, and API management.
* **React.js Official Documentation**. (n.d.). Retrieved from https://reactjs.org/docs/getting-started.html
* Offers insights into React.js, a front-end library used for creating interactive UIs, and explains React’s components, state management, and hooks.
* **Node.js Documentation**. (n.d.). Retrieved from https://nodejs.org/en/docs/
* Provides official documentation on Node.js, covering its asynchronous nature, server-side scripting, and the event-driven model.
* **Socket.io Documentation**. (n.d.). Retrieved from https://socket.io/docs/
* A guide to using Socket.io for real-time web applications, particularly focusing on the integration of real-time, bi-directional communication between clients and servers.
* **MERN Stack Tutorial**. (2021). Retrieved from https://www.freecodecamp.org/news/mern-stack-tutorial/
* A step-by-step guide on how to set up and use the MERN stack to create full-stack applications, covering MongoDB, Express, React, and Node.js integration.
* **Full-Stack React Projects**. (2020). Packt Publishing.
* A book that covers various full-stack development techniques using React, Node.js, and MongoDB, with practical examples and case studies.