WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

De-Link Chat Application using MERN stack and Socket.io

Kanika Tyagi[1], Deepali Singh[2], Bhumi Sharma[3]

RKGIT,Ghaziabad, 201003,India
RKGIT,Ghaziabad, 201003,India
RKGIT,Ghaziabad, 201003,India

ABSTRACT :

De-link is a real-time chat app developed using the MERN stack (MongoDB, Express.js, React.js, Node.js) and Socket.IO. It enables instant messaging, dynamic updates, and user-friendly interface. Utilizing MongoDB for database, Express.js for routing, React.js for UI, and Node.js for backend, Socket.IO facilitates bidirectional communication. Challenges like performance optimization and scalability were addressed. Features include user authentication, password encryption and group chats. Evaluation shows responsiveness, reliability, and usability.

Keywords: MongoDB, Express.js, React.js, Node.js, Mongoose, Bcrypt, npm, Chakra UI

1. Introduction :

Chat applications have become an integral part of modern communication, offering real-time interaction and collaboration among users across various platforms. However, developing a robust and efficient chat application poses its own set of challenges. To address these challenges and provide a seamless communication experience, we propose the development of a chat application using the MERN (MongoDB, Express.js, React.js, Node.js) stack along with Socket.io integration. This project aims to leverage the power of these technologies to create a scalable, real-time chat platform capable of handling a large volume of concurrent users while ensuring data integrity and security.

Traditional chat applications often rely on technologies that are not well-suited for real-time communication or struggle to scale effectively. By adopting the MERN stack, we can harness the flexibility and performance of JavaScript-based technologies throughout the development process. MongoDB offers a scalable and flexible database solution, allowing us to store and manage chat data efficiently. Express.js provides a robust framework for building server-side applications, enabling us to create a reliable backend API for handling user authentication, message routing, and other essential functionalities.

Moreover, integrating React.js on the client-side enables us to build a dynamic and responsive user interface, enhancing the user experience with features such as real-time message updates, typing indicators, and message previews. Node.js serves as the foundation of our chat application, facilitating event-driven, non-blocking I/O operations and enabling seamless communication between the client and server.

Incorporating Socket.io into our chat application further enhances its real-time capabilities by enabling bidirectional communication between the client and server over a WebSocket connection. This allows for instant message delivery, live updates, and presence detection, fostering a more interactive and engaging user experience.

The primary objective of this project is to develop a feature-rich chat application that combines the power of the MERN stack with Socket.io to deliver a seamless and scalable communication platform. By leveraging these technologies, we aim to address the limitations of traditional chat applications and provide users with an intuitive, real-time messaging experience. This research endeavor encompasses the design, development, and evaluation of the chat application, with a focus on performance, scalability, and user satisfaction.

The significance of this study lies in its potential to revolutionize the landscape of chat applications by offering a highly responsive and scalable platform that meets the evolving needs of users in today's digital age. By harnessing the capabilities of the MERN stack and Socket.io, we can overcome the challenges associated with traditional chat applications and deliver a superior communication experience. This paper will explore the design and implementation of the chat application, evaluate its performance and usability, and discuss its implications for future developments in real-time communication technology.

2. LITTERATURE REVIEW

Dr. Abhay Kasetwar, Ritik Gajbhiye, Gopal Papewar, Rohan Nikhare, Priya Warade [1] propose a real-time messaging system leveraging modern web technologies like JavaScript, React.js, and Node.js. Embracing the MERN stack, the project emphasizes developer productivity and user experience through open-source components. Key objectives include ensuring message security, enabling two-way communication, and facilitating unlimited data

transfer. The approach involves implementing basic functionalities utilizing network protocols like HTTP and WebSocket, with core features encompassing user registration, message editing, contact lists, and notifications. Future enhancements are envisioned to include video calls and voice recording, prioritizing data integrity and adaptability.

Akshata D Vhandale, Sayam N Gandhak, Saundarya A Karhale, Sandipkumar R Prasad, Prof. Sudhesh A Bachwani [2] focus on the development of a chat application with real-time capabilities and multi-platform functionality. Utilizing the MERN stack, including MongoDB, Express.js, React, and Node.js, the project aims to facilitate seamless communication among users. Key objectives include creating an intuitive graphical user interface (GUI) and ensuring platform independence to enhance communication, user-friendliness, and scalability for both organizational and personal use. Leveraging HTML, CSS, JavaScript, and additional technologies, the application targets efficient message transmission, user interaction, and content management, setting the groundwork for future enhancements and adaptations.

Yogesh Baiskar, Priyas Paulzagade, Krutik Koradia, Pramod Ingole, Dhiraj Shirbhate [3] explores modern web technologies, focusing on the MERN stack. It elucidates frontend development using HTML, CSS, and JavaScript, emphasizing frameworks like React.js. Backend development covers Node.js and Express.js, with MongoDB highlighted as the database choice. The MERN stack's advantages, including UI rendering efficiency and cost-effectiveness, are discussed alongside deployment considerations like Linux and hosting. Valuable insights into modern web development are provided, making it an essential resource for aspiring full-stack developers.

Rahul Khandelwal, Dishant Solanki, Teena Verma [4] proposes the development of a real-time chat application to address contemporary communication needs. Utilizing Node.js, Socket.IO, and MongoDB, the application aims for multi-platform responsiveness, enabling seamless messaging, file sharing, and location sharing. The proposed client-server architecture ensures bidirectional message transmission for real-time communication. Key features include user authentication, room creation, and intuitive interfaces. Emphasizing the significance of chat applications in modern communication, the research underscores the efficiency of Node.js and Socket.IO for development and deployment. Overall, the proposal targets enhancing communication efficacy through innovative technological solutions.

Manjeet Kumar, Vishal Thakur, Dheeraj Gurjar [5] research introduces a novel approach towards developing a Multi-User Web Chat Application utilizing Node.js and Socket.io. This endeavor aims to facilitate group communication within a minimalist framework, prioritizing user experience. Leveraging a Single-Page Application (SPA) architecture, the proposed system integrates Express for server initialization, HTML for rendering, and CSS for styling. Key functionalities encompass room creation, user verification, and real-time location sharing via Google Maps links. Future prospects entail enhancements such as voice and video messaging, file transfer capabilities, and personalized features, enriching the application's functionality and user engagement.

Shivansh Sethi [6] focuses on decentralization and user authentication for heightened security. Their approach leverages MongoDB, Express.js, React, and Node.js to facilitate seamless communication through private conversations and public chat rooms while enabling resource sharing. With an architecture comprising server-side components for managing client interactions and React components for user interface presentation, the system aims for scalability and flexibility. Future enhancements may include features like file transfer, voice and video messages, and group calling.

2.1. Problems in Existing Approaches

Dr. Abhay Kasetwar, Ritik Gajbhiye, Gopal Papewar, Rohan Nikhare, Priya Warade [1] emphasizes the development of a real-time messaging system using modern web technologies like JavaScript, React.js, and Node.js, following the MERN stack. However, the approach lacks thorough consideration of crucial aspects such as security measures, scalability, technical architecture, real-time communication efficiency, user experience, and cross-platform compatibility. While focusing on developer productivity and user experience, the paper overlooks the necessity of robust security measures and comprehensive technical architecture, potentially compromising user privacy and application performance.

Akshata D Vhandale, Sayam N Gandhak, Saundarya A Karhale, Sandipkumar R Prasad, Prof. Sudhesh A Bachwani [2] research paper centers on creating a real-time chat app with multi-platform support. However, it neglects scalability and integration with external systems. While emphasizing platform independence and user-friendliness, it lacks depth in user experience design. Additionally, comprehensive solutions for organizational communication needs are missing. This gap could hinder the app's effectiveness in handling larger user bases and diverse organizational requirements, potentially leading to usability issues and limited adoption. Addressing these concerns is crucial for ensuring the app's success in the broader context.

Yogesh Baiskar, Priyas Paulzagade, Krutik Koradia, Pramod Ingole, Dhiraj Shirbhate [3] provides a thorough overview of the components and functionalities of the MERN stack but neglects crucial aspects like scalability and security. While it briefly acknowledges scalability concerns, it lacks practical strategies to address them effectively. Additionally, the paper only briefly touches on cybersecurity, failing to delve into MERN-specific security considerations. This oversight leaves developers without comprehensive guidance on ensuring the scalability and security of MERN-based applications, highlighting a significant gap in the paper's coverage of essential development considerations.

Rahul Khandelwal, Dishant Solanki, Teena Verma [4] research paper focuses on designing and implementing a real-time chat application to provide users with a responsive, multi-platform communication tool capable of sharing messages, images, files, and location data. However, the existing approach fails to address critical issues such as scalability, security, data consistency, user experience design, error handling, and cross-platform compatibility. These shortcomings can hinder the application's effectiveness, leading to potential performance issues, security vulnerabilities, and a suboptimal user experience.

Manjeet Kumar, Vishal Thakur, Dheeraj Gurjar [5] provides a basic platform for real-time communication. However, the lack of extensive discussion on security measures like authentication and encryption leaves the application potentially vulnerable to threats. Without proper safeguards,

unauthorized access and data breaches could compromise user privacy and integrity of conversations. Enhancing security protocols is crucial to ensure the protection of user data and maintain trust in the application's reliability.

Shivansh Sethi [6] chat application's centralized architecture limits scalability and poses security risks. Although it recognizes decentralization trends, it lacks specific implementation strategies for resilience and masquerading attack prevention. Additionally, it primarily focuses on text communication, neglecting planned features like file transfer and voice/video messages. Vague outlines of testing methodologies and maintenance strategies raise concerns about long-term reliability and performance

3. DESIGNS AND IMPLEMENTATION

This study delved into the structure and design of a real-time collaboration program enabling multiple users to engage in synchronized file viewing and editing via a web browser.It outlines a comprehensive approach for developing a chat application using the MERN (MongoDB, Express.js, React.js, Node.js) stack. The process begins with meticulous planning and design, where the features, functionalities, and user interface of the application are defined and visualized through wireframes or mockups. Concurrently, the database schema is strategized to efficiently store user information, messages, and relevant data.

Moving on to the development environment setup, Node.js and npm are installed if not already available, followed by the establishment of a MongoDB database, either locally or through a cloud service like MongoDB Atlas. A MERN project is initialized, leveraging tools like Create React App for the frontend and Express Generator for the backend.

Backend development focuses on implementing user authentication, setting up API endpoints for user management, and integrating Socket.IO for real-time communication. Database integration with MongoDB involves defining schemas and implementing CRUD operations using Mongoose.

Frontend development revolves around designing and developing the user interface with React components, incorporating user authentication forms, and creating components for displaying chat messages and user lists. Real-time communication is facilitated through Socket.IO integration, enabling seamless interaction between clients and the server.

Thorough testing is conducted at various levels, encompassing unit tests for backend and frontend components, integration testing to ensure smooth communication, and testing of real-time functionality and scalability using tools like Postman or WebSocket testing tools.

Deployment involves hosting the application and database on platforms like Heroku, AWS, or DigitalOcean, coupled with the setup of CI/CD pipelines for automated testing and deployment. Environment variables are configured to safeguard sensitive information like database credentials and API keys.

Monitoring and maintenance are essential post-deployment activities, involving the implementation of logging and monitoring solutions to track performance and errors. Regular updates to dependencies and addressing security vulnerabilities ensure the application's robustness and reliability. User feedback is continuously gathered to drive improvements and guide future feature development.

4. . Implementation Result

Highlighting the implementation aspect, our application harnesses the power of the MERN stack, known for its extensibility. To address current limitations, we've developed a foundational real-time chat model with essential features including secure login and registration. Each page is explained in detail below in the images.

1) Registration and Login Page

First user has to register himself with name, email id and create password and then all entries will be saved in the database and password will be saved in encrypted form using npm library bcrypt as shown in figure 5 and at last he has to login himself with registered email id and password. For validation purpose an OTP will be send to the email id while registering.

Registration Page:

Username: input field = text
Email: input field = email
Password: input field = password

IF (Email == blank or password == blank):
    "Error message"
Else:
    "Registration Successful"

Fig.1. Register page Pseudo code

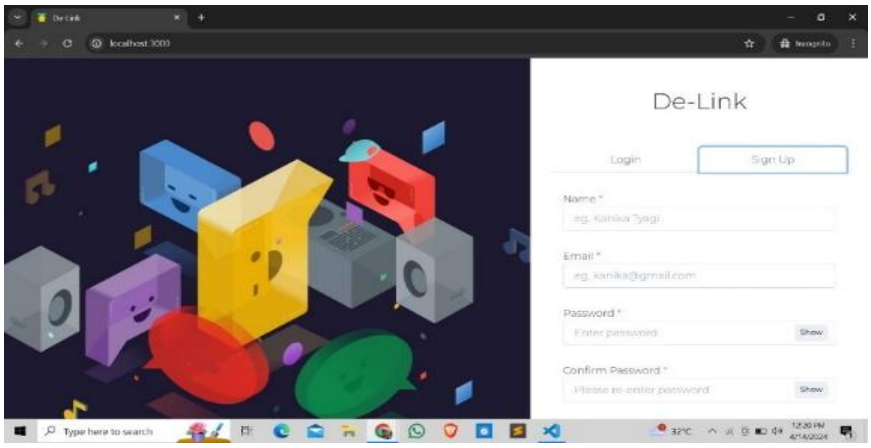

Fig.2. Registration page

Login Page:

Get login_id
Get password

IF (login_id = = Entered_username and password = = Entered_ password):
Login Successful
Redirects to Dashboard Screen

ELSE:
Login Failed

End

Fig. 3. Login page Pseudo code

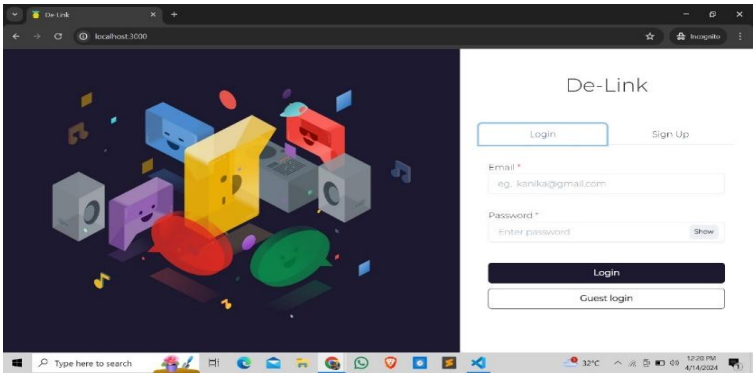

Fig. 4. Login Page

2) Dashboard

After logging in, users are taken to the dashboard, as shown in Figure 6. Here, they'll find different tabs representing various features of the app. In the middle of the screen, there's a message saying, " Select on a user to start chat " On the left side, users can see a list of friends they've added. At the top left,

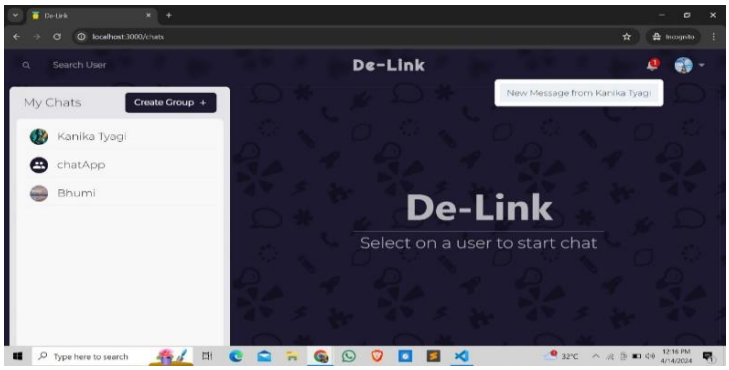

Fig. 5. Dashboard

3) Conversation

The web application facilitates private chatting between users, showcased in Figure 6. When engaging in a conversation, the chat box provides real-time notifications indicating when the other user is typing. As shown in the Figure 7 messages are organized chronologically by date and time. Additionally, each message is timestamped to indicate the time of sending or receiving. Similar to conventional messaging platforms, users can initiate and continue conversations with individuals on their friends list. However, to exchange messages, the recipient must be a confirmed friend; otherwise, communication attempts will be restricted.

5. Result and Discussion

Evaluate the effectiveness of Socket.IO in enabling real-time communication between users.Discuss the responsiveness of the chat application in delivering messages instantly to all connected clients. Test the application's scalability by simulating a large number of concurrent users. Measure how well the application handles increased traffic and maintains real-time functionality. Gather feedback from users regarding the ease of use and intuitiveness of the chat interface. Assess the overall satisfaction level of users interacting with the application. Evaluate the performance of the application in terms of loading times, responsiveness, and resource usage.

Use performance monitoring tools to identify any bottlenecks or areas for optimization. Review the security measures implemented, such as user authentication and data encryption. Perform security audits to identify and address potential vulnerabilities. Highlight any successful aspects of the chat application, such as seamless real-time communication, intuitive user interface, or efficient scalability. Discuss any positive feedback received from users during testing or after deployment. Identify challenges encountered during development, such as debugging real-time issues, optimizing performance, or integrating different components of the MERN stack. Reflect on lessons learned from building the chat application, including best practices, pitfalls to avoid, and areas for improvement in future projects. Share insights gained from working with the MERN stack and Socket.IO, including tips for optimizing performance and enhancing user experience. Propose potential enhancements or features to further improve the chat application.

Discuss ideas for future iterations, such as adding multimedia support, implementing message notifications, or integrating additional third-party services. Discuss the potential impact of the chat application and its relevance to various industries or use cases. Explore potential applications beyond traditional messaging, such as collaborative workspaces, customer support systems, or online communities. Encourage community engagement by open-sourcing the codebase, sharing tutorials or documentation, and inviting contributions from other developers. Foster a supportive community around the chat application to encourage collaboration and innovation. (1)

4. Conclusion and Future Work

In conclusion, developing a chat application using the MERN stack has been a challenging but rewarding experience. The use of MongoDB, Express, React, and Node.js provides a powerful and flexible framework for creating real-time communication and collaboration solutions that can be tailored to meet a wide range of use cases and industries. The app has been designed with user experience in mind, and features such as real-time message updates and user a authentication have been implemented to provide a seamless and secure communication experience. The scalability and robustness of the MERN stack ensure that the app can handle a high volume of users and messages without compromising on performance. Going forward, there are many opportunities for further development and improvement of the app. This includes adding new features such as video and voice chat, integrating with other applications and platforms, and enhancing the user interface to make it more intuitive and user-friendly. Overall, the chat app developed using the MERN stack represents a significant achievement in the field of real-time communication and collaboration, and has the potential to revolutionize the way people connect and communicate online.

Implement features to improve user engagement and interaction, such as emojis, reactions, or message threading. Introduce message delivery status indicators (e.g., read receipts, message sent/seen timestamps). Develop a responsive design to ensure optimal user experience across various devices and screen sizes. Extend the chat application to support multimedia content, such as image, audio, and video sharing. Implement file upload functionality for users to share documents or media files within the chat interface. Integrate third-party APIs or services for media processing and storage. Enhance the search functionality to allow users to search for specific messages or conversation.

REFERENCES :

[1] Dr.Abhay Kasetwar ,Ritik Gajbhiye ,Gopal Papewar ,Rohan Nikhare ,Priya Warade,
“Development of Chat Application”.
[2] Akshata D Vhandale, Sayam N Gandhak, Saundarya A Karhale, Sandipkumar R Prasad,
Prof. Sudhesh A Bachwani “ An overview of real-time chat application ” .
[3] Yogesh Baiskar, Priyas Paulzagade, Krutik Koradia, Pramod Ingole, Dhiraj Shirbhate “
MERN: A Full-Stack Development ” .
[4] Rahul Khandelwal, Dishant Solanki, Teena Verma “ Design & Implementation of real time
chat application ” .
[5] Manjeet Kumar, Vishal Thakur ,Dheeraj Gurjar “ Multi-User Web Chat Application using
Node.js and Socket.io”.
[6] Nita Thakare, Nitin Deshmukh, Anshul Vairagade, AyushNagarare, Himanshu Kamane, Rajat Mohod (2022).”Real Time Chatting Web Application”.
[7] “simple peer”[Online].
Available: https://www.npmjs.com/package/ simple-peer.
[8] ”WebRTC based peer to peer voice,video calling and messaging web app build with MERN stack,” [Online].
Available: https://github.com/saalikmubeen/talkhouse.
[9] ”What is Material UI in React?” ,[Online]. Available: https://www.educative.io/answers/what-is-material-ui-inreact
[10] ”How To Manage State in React with Redux” Online].
[1] Dr.Abhay Kasetwar ,Ritik Gajbhiye ,Gopal Papewar ,Rohan Nikhare ,Priya Warade,
“Development of Chat Application”.
[2] Akshata D Vhandale, Sayam N Gandhak, Saundarya A Karhale, Sandipkumar R Prasad,
Prof. Sudhesh A Bachwani “ An overview of real-time chat application ” .
[3] Yogesh Baiskar, Priyas Paulzagade, Krutik Koradia, Pramod Ingole, Dhiraj Shirbhate “
MERN: A Full-Stack Development ” .
[4] Rahul Khandelwal, Dishant Solanki, Teena Verma “ Design & Implementation of real time
chat application ” .
[5] Manjeet Kumar, Vishal Thakur ,Dheeraj Gurjar “ Multi-User Web Chat Application using
Node.js and Socket.io”.
[6] Nita Thakare, Nitin Deshmukh, Anshul Vairagade, AyushNagarare, Himanshu Kamane, Rajat Mohod (2022).”Real Time Chatting Web Application”.
[7] “simple peer”[Online].
Available: https://www.npmjs.com/package/ simple-peer.
[8] ”WebRTC based peer to peer voice,video calling and messaging web app build with MERN stack,” [Online].
Available: https://github.com/saalikmubeen/talkhouse.
[9] ”What is Material UI in React?” ,[Online]. Available: https://www.educative.io/answers/what-is-material-ui-inreact
[10] ”How To Manage State in React with Redux” Online].