International Journal of Research Publication and Reviews

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

AI-Enhanced Code Share & Messaging Platform

Damini Sahu[1], Shreeya Sahu[1], Sakshi Randive[1] Sampada Massey[2]

Department of CSE(AIML), Shri Shankaracharya Technical Campus Junwani Bhilai (C.G.), India
[1]sahudamini5460@gmail.com [1]shreeyasahu786@gmail.com [1]sakshirandive314@gmail.com [2]sampada.massey@gmail.com

A B S T R A C T:

The emergence of new technologies has brought about significant changes in the way people communicate with each other. One of the most popular ways of communication in today's digital age is through messaging applications. To facilitate this need, With the growing demand for intelligent conversational systems, the integration of large language models (LLMs) into messaging platforms has become increasingly relevant. In our project, we propose an innovative enhancement to the existing MERN-based real-time chat application by integrating the Gemini API, enabling AI-powered interactions within chat rooms. This approach aims to emulate the functionality seen in modern messaging applications such as WhatsApp's @Meta AI, providing users with contextual responses, automated assistance, and real-time information retrieval directly within the chat interface.

INTRODUCTION :

Chat applications have become an integral part of our day-to-day life and have had a significant impact on how we communicate with each other. With numerous chat applications available in the market, each offering unique features and capabilities, users are spoilt for choice. Companies that develop these applications compete with each other to add new features and improve the user experience with each release.
However, with the growing concern of data theft, companies must ensure the security of their users' data and protect them from third-party data breaches. To address this, the basic chatting system should involve both sending and receiving processes simultaneously, which can be achieved through the MERN concept.
Developers worldwide are constantly striving to enhance the user experience of
chat applications and improve their workflow to deliver projects and changes quickly. This is where stacks come into play, which allow developers to build web applications quickly and efficiently. MERN stack is popular and built on JavaScript that offer an end-to- end framework for building comprehensive web apps that enable browsers to connect with databases.
This research paper will give idea and features of the project, In the current era of remote work and online education,
platforms that support real-time collaboration have gained significant traction. Traditional coding environments often lack seamless communication features, while chat platforms are not optimized for programming tasks. This project addresses these gaps by combining a live code editor, real-time chat, and AI assistance into a single unified platform. By leveraging the Gemini API, we introduce AI capabilities that can assist with programming queries, documentation, and debugging support directly within the collaboration space.

LITERATURE REVIEW

To develop a chat application, the first step was to design a database schema that could store user information, chat rooms, and messages. MongoDB, a document-oriented database, was chosen for this purpose due to its flexibility and ease of use. The schema was designed to have multiple collections, each responsible for storing different types of data.

Once the database schema was in place, the next step was to build the server-side API that would handle requests from the client- side application. Express.js, a fast and flexible Node.js web application framework, was used to build the server- side API. It provides features such as routing, middleware, and templating, making it an ideal choice for building web applications and APIs.

On the client-side, ReactJS was chosen to create a responsive and intuitive user interface. ReactJS is a JavaScript-based framework that simplifies the development of web applications by providing a structured framework for creating dynamic views. It allows developers to create complex user interfaces with ease, using reusable code and components.

User authentication was also an important feature of the chat application, and this was implemented using JSON Web Tokens (JWT). JWT is a secure and easy-to- use authentication mechanism that allows users to securely transmit information between parties.

Finally, Node.js was used to deploy the application on a cloud hosting service. Node.js is a platform built on the Chrome V8 JavaScript engine that allows developers to run JavaScript code outside of a web browser on the server side, providing a powerful and efficient way to build server- side applications using JavaScript. It allows Users to run JavaScript code on the server. It is fast and scalable, making it ideal for building web applications that can handle high traffic.

In conclusion, developing an application requires a combination of technologies and frameworks, each serving a specific purpose. The choice of technologies depends on the specific requirements of the application, such as real-time message updates, user authentication, and scalability. MongoDB, Express.js, ReactJS, socket.io, JWT, and Node.js were the technologies chosen for this particular chat application, resulting in a robust and feature-rich application that met the requirements of the project.

Here is a more detailed explanation of the technologies used in this web-application:

1. HTML, CSS, and JavaScript form the backbone of web development and are essential tools for creating dynamic and interactive websites. HTML is used to structure and define the content of web pages, CSS is used for styling and layout, and JavaScript is used for creating dynamic interactions and functionality.

2. **MongoDB:** It is a cross-platform document-oriented NoSQL database. MongoDB stores data in JSON-like documents, making it easy to work with for developers. It provides support for ad-hoc queries, indexing, and aggregation, making it a popular choice for building web and mobile applications that require a flexible and scalable database.
3. **Express:** Express is a web application framework for Node.js that provides developers with a powerful set of tools to build back-end web applications and APIs. It includes features such as routing and middleware, which make it easier to handle HTTP requests and responses.
4. **React:** React is based on a component- based architecture, which MERN developers can create reusable UI components that can be used across an application. React also manages the state of an application more efficiently, making it easier to build complex applications.
5. **Node.js:** Node.js is a JavaScript runtime environment that enables developers to execute JavaScript code on the server- side. It is cross-platform and open- source, which allows for the creation of scalable and high- performance web applications and APIs. Node.js is known for its speedy and efficient event-driven, non- blocking I/O model.

To enhance the collaborative development experience, we integrated a VS Code editor into the platform using the Monaco Editor, which offers syntax highlighting, autocompletion, and language support similar to Visual Studio Code. For executing code within the browser itself, we utilized the Web Container API, enabling a full Node.js environment to run directly in the user's browser without any backend servers. The output of executed code is rendered securely inside an iframe sandbox, providing a safe and isolated environment. This setup not only simulates a real development environment but also supports real-time file management and execution, allowing users to write, run, and test code collaboratively—all within a single web interface.

The MERN stack combines these technologies to create a comprehensive web application framework. MongoDB provides a flexible and scalable database, Express provides a simple and minimalist back-end framework, React provides a powerful front-end library for building user interfaces, and Node.js used to run JavaScript code on the server-side. Together, these technologies provide a full-stack solution for building robust and scalable web applications.

PROPOSED SCHEME

This section explains the development plan for this web application:
The development plan for our chat application involved creating a platform for users to communicate with each other in real-time using the MERN stack, which is a widely used and modern technology stack in the industry. The application's frontend was developed using React, while the backend was developed using Node.js and Express.js, and the data was stored in MongoDB.

In the first phase, we focused on setting up the development environment and installing the necessary tools and libraries. We created a project skeleton and defined the basic structure of our application. This phase was critical in establishing a solid foundation for our development process.

In the second phase, Traditional chat applications enable peer-to-peer or group communication without intelligent context awareness. However, the recent advancements in LLMs present a new opportunity to elevate user experience by embedding AI-driven conversational agents. This paper explores the integration of the Gemini API into a MERN-based real-time chat system to allow users to summon AI assistance by simply mentioning "@ai" within the chat interface, we designed the database schema and created the necessary models and controllers. We also implemented user authentication and authorization.

In the final phase, to enhance the collaborative development experience, our platform integrates a VS Code-like code editor providing features such as syntax highlighting, autocompletion, and support for multiple programming languages. For seamless in-browser code execution, leverage the Web Container API, which simulates a full Node.js environment directly within the user's browser, eliminating the need for backend servers. The execution results are securely displayed within an iframe sandbox, ensuring a safe and isolated environment. This setup allows users to collaboratively write, run, and test code in real-time, simulating a true development environment while offering seamless file management and execution within a single web interface.

We conducted extensive testing and debugging to ensure that our application was free of errors and performed optimally. We also deployed the application to a cloud platform, such as AWS or Heroku, to make it accessible to users.

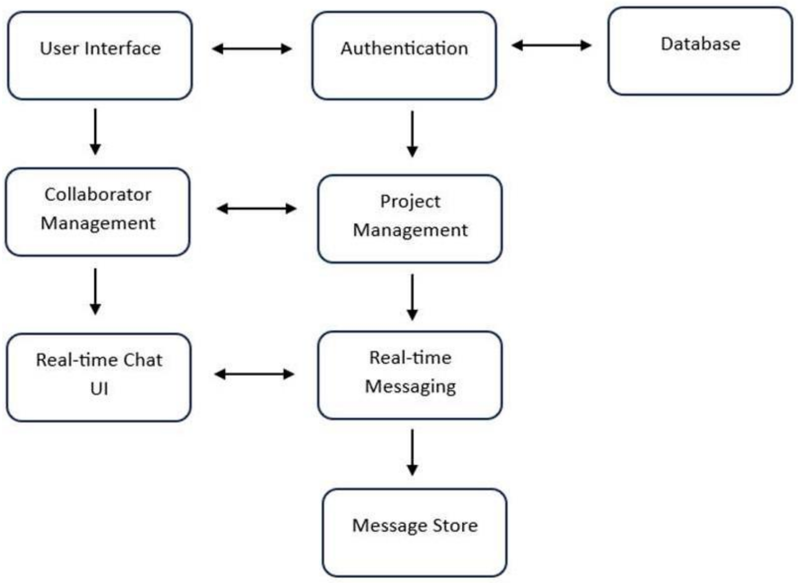


Fig 3.1: Data flow diagram of Real-Time Chat

The flow of working of a chat application built using the Mern stack can be explained as follows:

User registration and authentication:

The first step in using the chat application is to create an account. The user is required to enter their details and register to access the application. Once registered, the user can log in to the application using their email and password. Joining chat rooms: The next step is to join a chat room. The user can either create a new chat room or join an existing one by entering the room name or code. Once the user has joined a chat room, they can start chatting with other users who are also part of that chat room. Sending and receiving messages: The core feature of the chat application is the ability to send and receive messages in real-time. Using web sockets, the application enables users to send and receive messages instantly, allowing for a seamless conversation experience.

User authentication and authorization: To ensure data security and user privacy, the application incorporates user authentication and authorization features. Users must log in to access the application and only authorized users can join specific chat rooms.

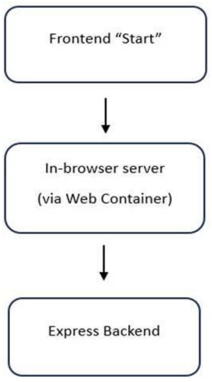


Fig 3.2: Data flow diagram of Server Management (via Web Container)

The backend server is built with Express, using middleware for request handling and error catching. A "Start Server" button in the frontend triggers a Web Container environment that mounts backend files and runs the server entirely in the browser, eliminating the need for external hosting.

RESULTS AND DISCUSSION

Following is some of the results from our application:
This is the signup page where users have to enter their details like name, email address, password and also can upload your pictures to sign up our applications.

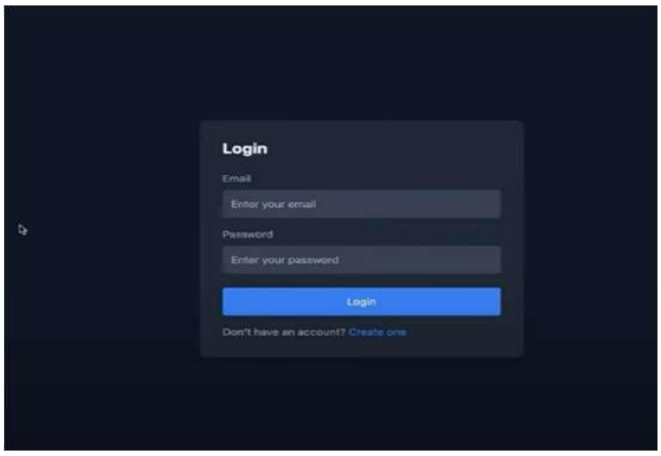


Fig 4.1: Sign-up Page

See as many characters as the user is logged in and our data will be reflected in each authorized user.

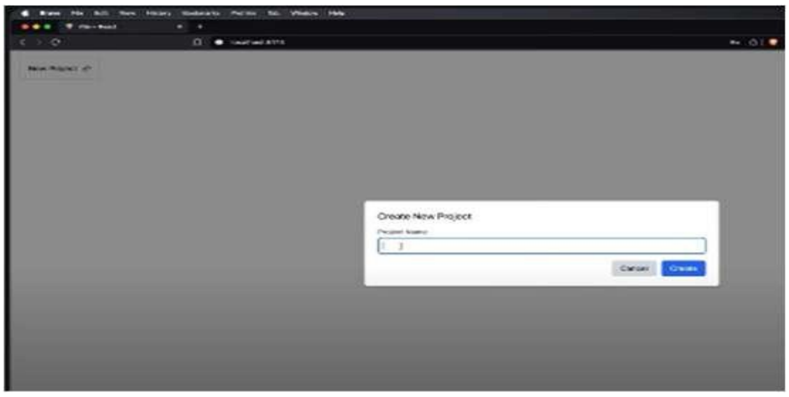


Fig 4.2: Collaborator

The application interface allows users to communicate with any registered user directly, enabling seamless communication and fostering community engagement.

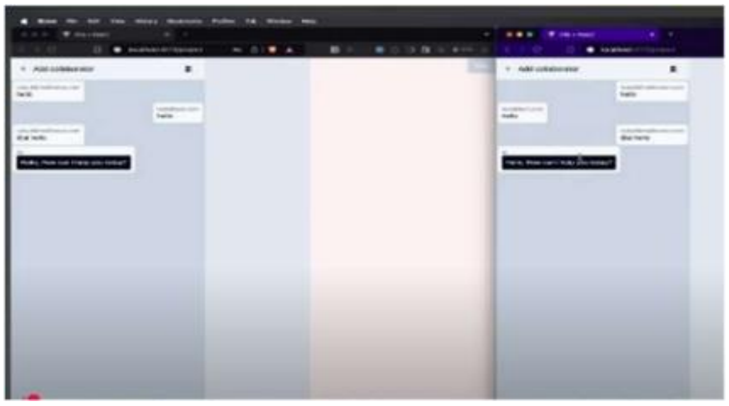
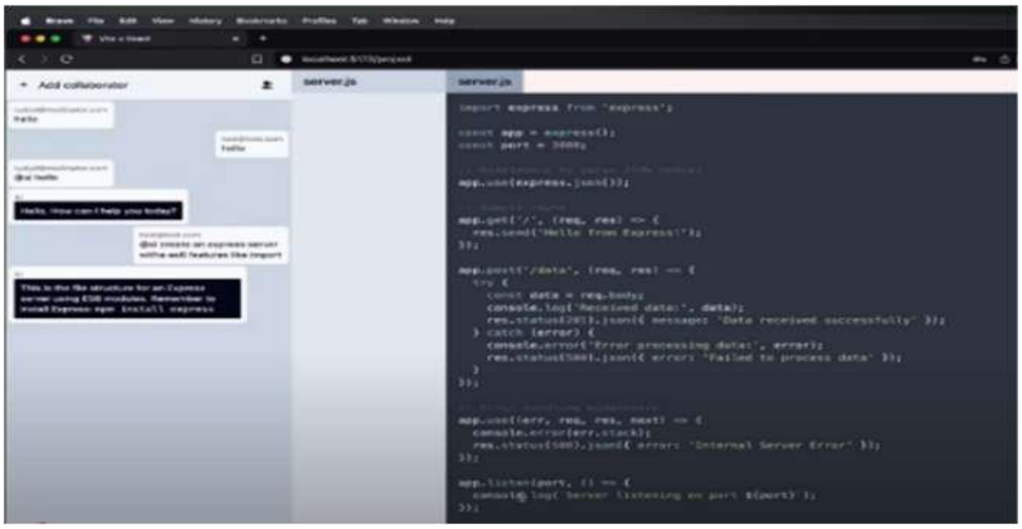


Fig 4.3: Chatting Interface

The backend uses Express to manage how the app works and handles errors. There's a button that, when clicked, starts the server inside the browser using we containers, so the app runs fully in the browser without needing any online server or hosting.



hello

hello

@ai hello

Hello, How can I help you today?

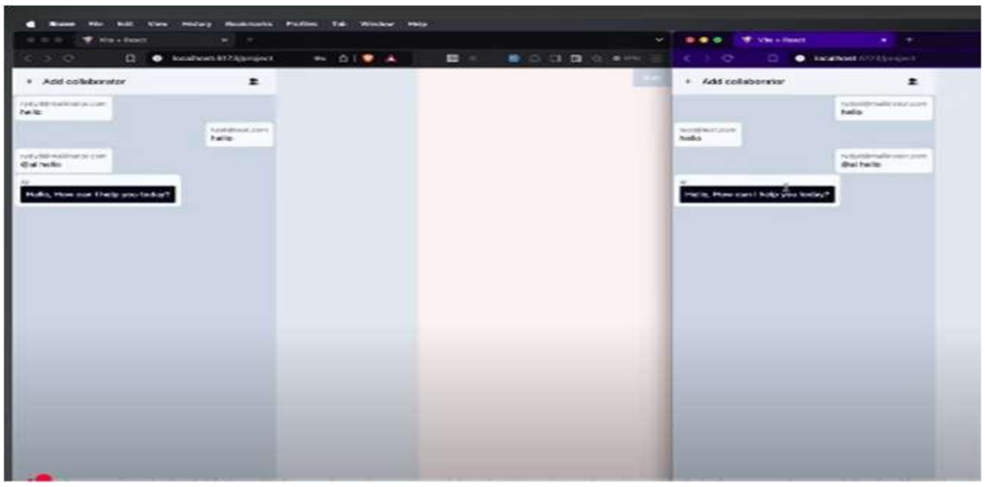
@ai create an express server withe es6 features like import

This is the file structure for an Express server using ES6 modules. Remember to install Express: npm install express

```
import express from 'express';

const app = express();
const port = 3000;

// Middleware to parse JSON bodies
app.use(express.json());

// Sample route
app.get('/', (req, res) => {
  res.send('Hello from Express!');
});

app.post('/data', (req, res) => {
  try {
    const data = req.body;
    console.log('Received data:', data);
    res.status(201).json({ message: 'Data received successfully' });
  } catch (error) {
    console.error('Error processing data:', error);
    res.status(500).json({ error: 'Failed to process data' });
  }
});

// Error handling middleware
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: 'Internal Server Error' });
});

app.listen(port, () => {
  console.log(`Server listening on port ${port}`);
});
```

Fig 4.4: Overall Interface

The application interface allows users to communicate with any registered user directly, enabling seamless communication and fostering community engagement.



hello

hello

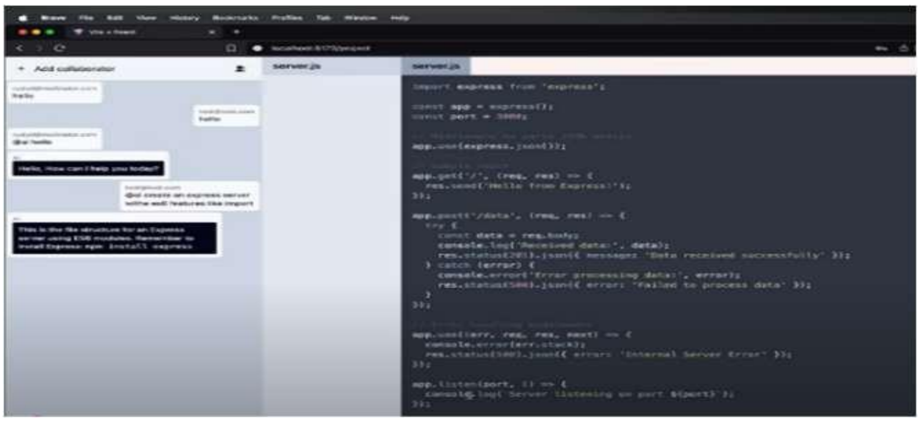
@ai hello

Hello, How can I help you today?

Fig 4.3: Chatting Interface

The backend uses Express to manage how the app works and handles errors. There's a button that, when clicked, starts the server inside the browser using we containers, so the app runs fully in the browser without needing any online server or hosting.

.


hello

hello

@ai hello

Hello, How can I help you today?

@ai create an express server withe es6 features like import

This is the file structure for an Express server using ES6 modules. Remember to install Express: npm install express

```
import express from 'express';

const app = express();
const port = 3000;

// Middleware to parse JSON bodies
app.use(express.json());

// Sample route
app.get('/', (req, res) => {
  res.send('Hello from Express!');
});

app.post('/data', (req, res) => {
  try {
    const data = req.body;
    console.log('Received data:', data);
    res.status(201).json({ message: 'Data received successfully' });
  } catch (error) {
    console.error('Error processing data:', error);
    res.status(500).json({ error: 'Failed to process data' });
  }
});

// Error handling middleware
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: 'Internal Server Error' });
});

app.listen(port, () => {
  console.log(`Server listening on port ${port}`);
});
```

Fig 4.4: Overall Interface

CONCLUSION

In conclusion, the development of this real-time collaborative platform successfully demonstrates the integration of modern web technologies with AI- driven assistance to enhance both communication and coding productivity. By leveraging the MERN stack, Socket.IO, and the Web Container API, we created a seamless environment where users can chat, write code collaboratively, and execute it directly within the browser. The integration of Google's Gemini API further elevates the platform by providing contextual AI support, enabling users to receive intelligent suggestions, code explanations, and real-time feedback during their workflow. This fusion of chat, code editing, live execution, and AI not only streamlines collaborative development but also encourages learning, experimentation, and innovation in a unified digital workspace. The project serves as a scalable foundation for future advancements in AI-assisted software development and collaborative platforms. 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.
As a result, the application supports real- time collaboration for up to 10 users simultaneously within a single project, enabling efficient teamwork through integrated chat and code editing features. To use the application, first register in the web-application and to use AI write prompt with @ai then your query. To run the code use RUN button on the Right side in the UI.

REFERENCES :

1. Masiello Eric. Mastering React Native. January 11; 2017. This book is a comprehensive guide to building mobile applications using React Native.
2. Naimul Islam Naim. ReactJS: An Open-Source JavaScript library for front-end development. Metropolia University of Applied Sciences. This article provides an overview of ReactJS and its key features for front-end web development.
3. Stefanov Stoyan, editor. React: Up and Running: Building web Applications. First Edition; 2016. This book is a beginner-friendly introduction to React, covering its core concepts and providing practical examples for building web applications.
4. Horton Adam, Vice Ryan. Mastering React; February 23; 2016. This book provides a comprehensive guide to React, covering its core concepts,
   practical examples, and advanced techniques for building complex applications.
5. Alex Kondov. Express Architecture Review. This article provides a review of the architecture of Express.js, a popular web framework for building Node.js applications.
6. Express.js documentation. This documentation provides a comprehensive guide to building web applications using Express.js.
7. Adam Horton. Node.js vs Python: What to Choose. This article provides a comparison of Node.js and Python for web development, highlighting their strengths and weaknesses.
8. Node.js documentation. This documentation provides a comprehensive guide to building server- side applications using Node.js.