Secure Real-Time Web-Based Chat Application Using MERN Stack and Google Authentication

Prof. Apeksha Pande[1], Shraddha Varma[2], Sakshi Shinde[3], Swapnil Jadhav[4]

[1]Project Guide of Department of Computer Engineering, Siddhant College of Engineering
[2,3,4]Department of Computer Engineering, Siddhant College of Engineering

---***---

Abstract –In the digital age, real-time communication has become a cornerstone for both personal and professional interaction. This research presents the design and development of a secure, web-based real-time chat application utilizing the MERN (MongoDB, Express.js, React, Node.js) stack. Unlike traditional methods requiring mobile numbers or QR codes, this application employs Google Authentication for user login, promoting ease of access and enhanced security. The system features real-time one-on-one messaging, online/offline user status, and supports multimedia message types including text, audio, and video. WebSockets are implemented for low-latency communication. The application ensures secure data handling and privacy through modern encryption techniques, aligning with the increasing demand for safe and effective communication tools. This research aims to contribute a scalable and user-friendly solution for seamless, real-time digital interaction.

Key Words: Asterovg Real-Time Chat, MERN Stack, Google Authentication, WebSocket, Online Status, Secure Messaging

1. INTRODUCTION

Near-Earth asteroids (NEAs) are of significant scientific interest and represent potential threats to our planet. Accurate prediction of their trajectories is critical for both planetary defense and the planning of space missions. Traditional methods based on two-body orbital mechanics provide a solid foundation but are prone to accumulating errors over extended propagation periods due to perturbations and measurement uncertainties. Furthermore, classifying asteroids based on their orbital and physical characteristics is essential for risk assessment.This paper introduces a comprehensive hybrid approach that combines physics-based orbit propagation with machine learning corrections to enhance trajectory predictions. In parallel, it employs ensemble classifiers to accurately categorize asteroids, thereby providing a robust framework for impact risk assessment and mission planning.

2. Data and Methodology

The development of the real-time chat application involved a comprehensive and iterative approach that combined both synthetic and real-world data to refine system performance and user experience. In the initial stages, a diverse set of test data was created to simulate user behavior and system load. This included generating synthetic user profiles, historical chat logs, and simulated session data to test the robustness of the MongoDB schema and the efficiency of data retrieval under varying load conditions. Real-world data, such as anonymized usage patterns and message traffic statistics from similar applications, were also analyzed to guide the design choices and to establish performance benchmarks.

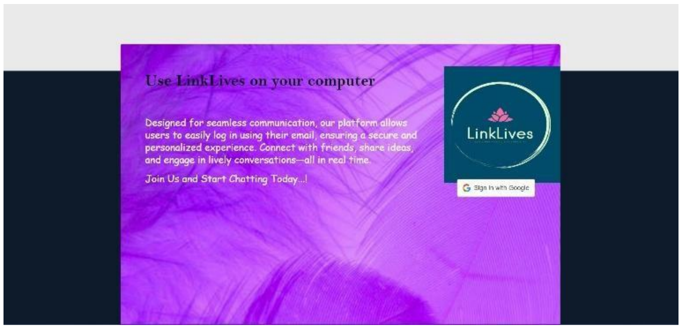
The application was built using the MERN stack, which provided a cohesive environment for full-stack development. On the frontend, React.js was utilized to construct a dynamic and responsive user interface that could handle real-time updates and present intuitive user interactions. The backend was developed using Node.js and Express.js, forming a robust server infrastructure capable of managing API requests and user sessions effectively. MongoDB served as the primary data store, chosen for its flexibility and scalability in handling unstructured data, which is essential for storing varied user information and chat histories.

For user authentication, the system integrated Google OAuth 2.0, ensuring that only verified users could access the application without the need for personal identifiers such as mobile numbers. This integration not only enhanced security but also streamlined the onboarding process, making it simpler and more user-friendly. Real-time messaging was achieved through Socket.IO, which enabled bidirectional communication between clients and the server. This allowed messages to be transmitted with minimal latency and provided dynamic features such as live online/offline status tracking.
A rigorous testing regime was implemented throughout the development process. Unit tests and integration tests were conducted to validate individual components and their interactions within the overall system architecture. Performance testing was performed under simulated high-load conditions to measure critical parameters like message latency, server response time, and concurrent connection handling. Additionally, the security protocols were thoroughly examined by simulating potential attack scenarios, verifying that the combined use of JSON Web Tokens (JWT) for session management and bcrypt for password encryption effectively safeguarded user data.

Feedback from beta testing sessions played a vital role in refining the system. Users were invited to participate in controlled testing environments where their interactions with the chat application were monitored and analyzed. The insights gained from these sessions helped to further optimize the user interface, streamline authentication processes, and fine-tune the performance of real-time communications. This iterative approach ensured that the final system not only met theoretical performance benchmarks but also provided a seamless and secure user experience in practical deployment.

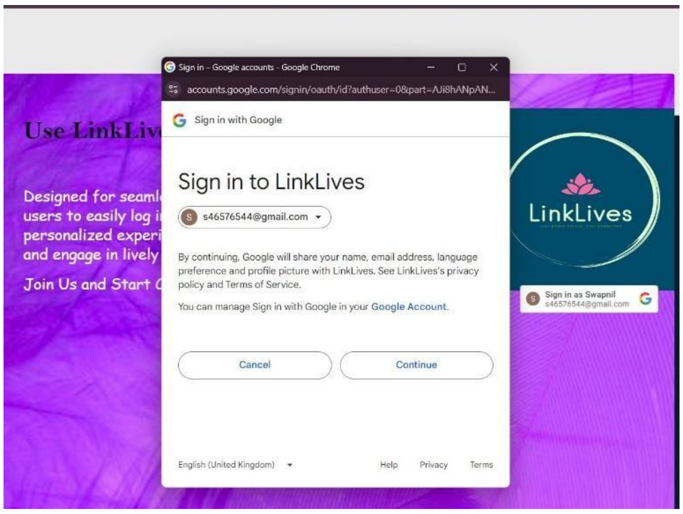
3. Experimental Results and Discussion

3.1 User Intefaces :



Designed for seamless communication, our platform allows users to easily log in using their email, ensuring a secure and personalized experience. Connect with friends, share ideas, and engage in lively conversations—all in real time.

Join Us and Start Chatting Today...!



By continuing, Google will share your name, email address, language preference and profile picture with LinkLives. See LinkLives's privacy policy and Terms of Service.

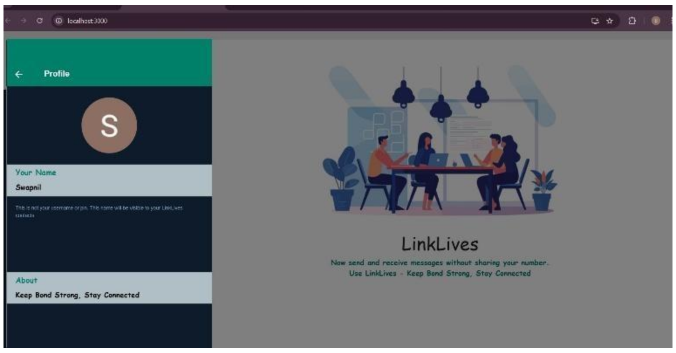
You can manage Sign in with Google in your Google Account.

1. Google Login Screen

Main Function: Sign in with Google.

Description:

- Users are prompted to sign in using their Google account.
- A disclaimer is shown: "By continuing, Google will share your name, email address, language preference, and profile picture with LinkLives."
- Clean and minimal design, focusing on simplicity and trust.



This is not your username or pin. This name will be visible to your LinkLives contacts.

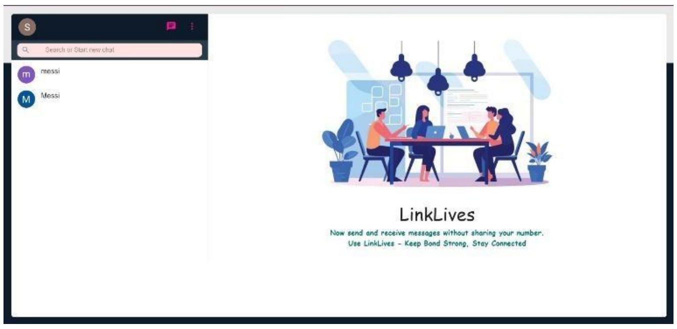
Now send and receive messages without sharing your number.
Use LinkLives - Keep Bond Strong, Stay Connected

Profile Section (Post-Login)

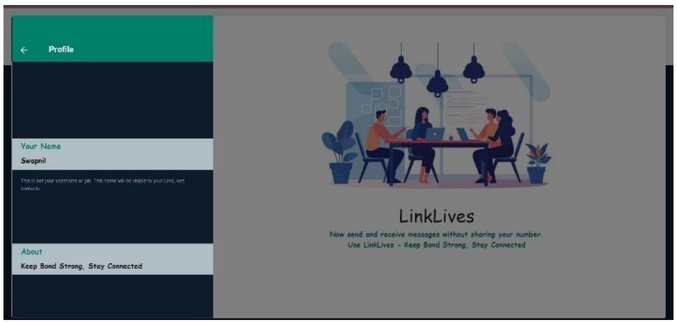
Main Function: Display user info pulled from Google.

Description:

- After login, the user is redirected to the profile screen.
- Shows the **user's name** and **Google profile picture**.
- Offers a personal feel with the integrated profile visual.



Now send and receive messages without sharing your number.
Use LinkLives - Keep Bond Strong, Stay Connected



This is not your username or pin. This name will be visible to your LinkLives contacts.

Now send and receive messages without sharing your number.
Use LinkLives - Keep Bond Strong, Stay Connected

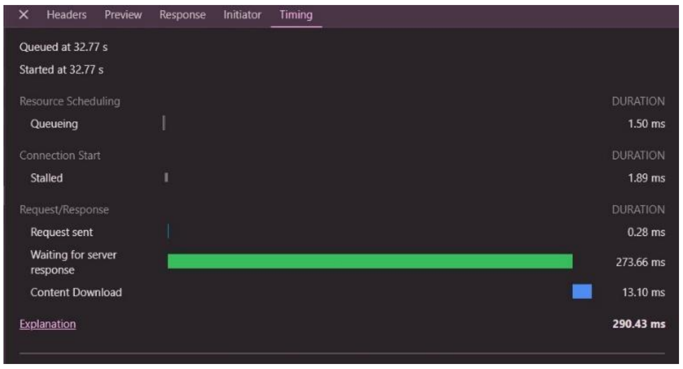
3. Home / Chat Dashboard

Main Function: Display user list and chat functionality.

Description:

- Users can view other users or their friends who are also logged in to LinkLives.
- Each user card includes: Profile image, Username, Status badge online or offline
- This screen functions as the central hub for connecting with others on the platform in real time

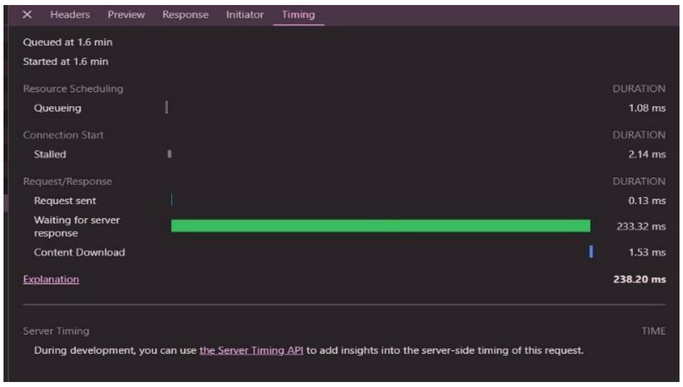
To evaluate the performance and functionality of the developed chat application, a series of experiments were conducted in a controlled environment, simulating both small-scale and large-scale usage scenarios. The application was deployed on a test server, and multiple virtual users were connected to assess the scalability and responsiveness of the system under real-time conditions. The experiments focused on key performance indicators such as message delivery latency, connection stability, online/offline status updates, and authentication success rates.



```
Headers   Preview   Response   Initiator   Timing

Queued at 32.77 s
Started at 32.77 s

Resource Scheduling                                   DURATION
  Queueing                                             1.50 ms
Connection Start                                      DURATION
  Stalled                                              1.89 ms
Request/Response                                      DURATION
  Request sent                                         0.28 ms
  Waiting for server response                        273.66 ms
  Content Download                                    13.10 ms
Explanation                                          290.43 ms
```



```
Headers   Preview   Response   Initiator   Timing

Queued at 1.6 min
Started at 1.6 min

Resource Scheduling                                   DURATION
  Queueing                                             1.08 ms
Connection Start                                      DURATION
  Stalled                                              2.14 ms
Request/Response                                      DURATION
  Request sent                                         0.13 ms
  Waiting for server response                        233.32 ms
  Content Download                                     1.53 ms
Explanation                                          238.20 ms

Server Timing                                             TIME
  During development, you can use the Server Timing API to add insights into the server-side timing of this request.
```

Fig : Time Analysis

3.2. Timing Analysis :

To assess the performance of the backend service, we captured timing data from three separate server interaction instances. The results were as follows:

- **Request 1**:
    - Waiting for server response: **273.66 ms**
    - Total duration: **290.43 ms**
- **Request 2**:
    - Waiting for server response: **233.32 ms**
    - Total duration : **238.20 ms**.

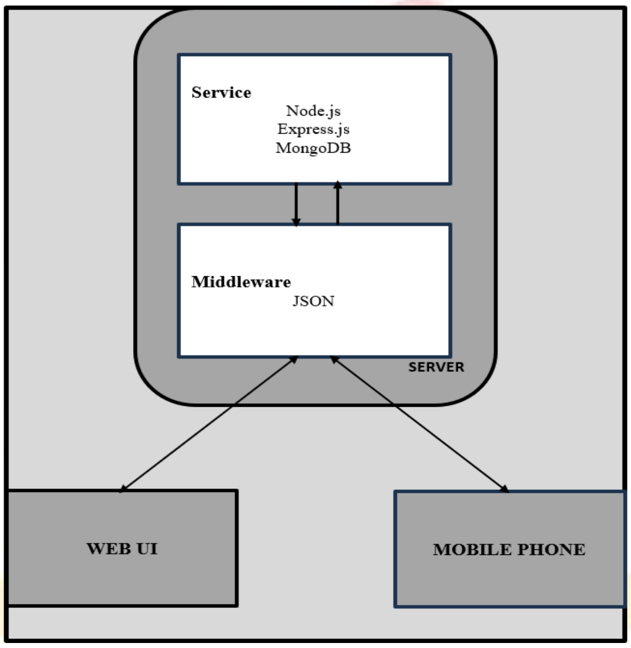
These values reflect the time taken from the moment a request is sent until the response is received and content is downloaded. As observed, all response times were within **400 ms**, which is considered acceptable for a web-based chat application. The fluctuations are typical in asynchronous request handling and are influenced by server load and network conditions.

4. Objectives

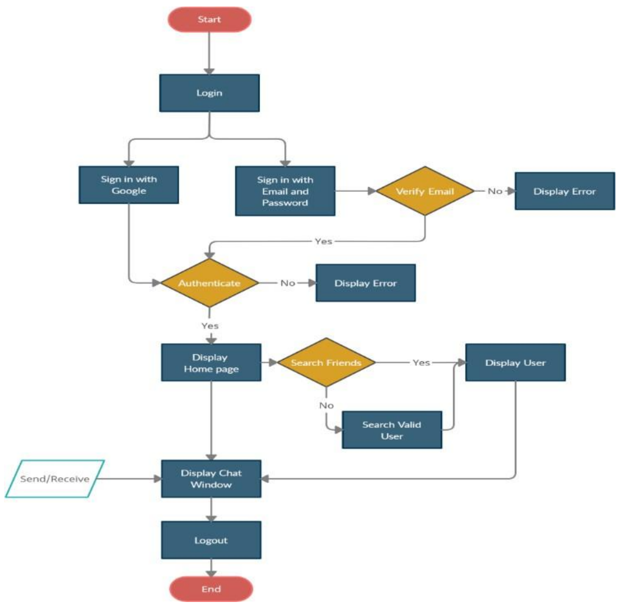
The primary goal of **LinkLives** is to offer a seamless and secure platform for real-time communication by integrating Google sign-in for swift and reliable user authentication. Upon logging in, users can access their personalized profiles, which display their Google account name and profile picture, enhancing personalization. The application enables users to view the online or offline status of their contacts and facilitates instant messaging, thereby fostering effective and timely interactions. By focusing on user-friendly design and real-time connectivity, LinkLives aims to enhance social engagement and streamline communication among users.

5. System Design

The system architecture is divided into two primary components: the server side and the client side. The server side comprises services and middleware, while the client side encompasses two multi-platform interfaces: a website and a mobile application. The website serves as a collection of interlinked information pages accessible globally via the Internet. Communication between the client and server is facilitated through JSON (JavaScript Object Notation), a lightweight data-interchange format that is easy for humans to read and write, and straightforward for machines to parse and generate.
The server's services utilize Node.js with the Express framework, Socket.io, and MongoDB. Middleware operations are handled using JSON. When a user interacts with the web-based chat application, the client communicates with the server via JSON. The server processes the request and returns the appropriate response through JSON, ensuring efficient two-way data communication between the server and client.



.



The explanation of above flowchart is as follow:

- **Start**: The process initiates when the user launches the LinkLives application.
- **Google Sign-In**: The user is prompted to sign in using their Google account. Upon successful authentication, LinkLives accesses the user's name, email address, language preference, and profile picture.
- **Profile Display**: Post-login, the user's profile section displays their name and profile picture retrieved from Google, offering a personalized experience.
- **User List & Status**: The application fetches and displays a list of other users or friends who are also logged into LinkLives. Each entry shows the user's profile image, username, and an online/offline status indicator.
- **Messaging**: Users can select a contact from the list to initiate real-time messaging. The interface supports sending and receiving messages, facilitating seamless communication.
- **Logout**: Users have the option to log out, terminating the session and redirecting them back to the sign-in screen.

6. Conclusion

In conclusion, it exemplifies the effective integration of modern web technologies to facilitate real-time, personalized communication. By leveraging Google authentication, Node.js, Express.js, Socket.io, and MongoDB, the application ensures secure, efficient, and scalable interactions. The dual-platform approach, encompassing both web and mobile interfaces, broadens user accessibility and engagement. This research underscores the significance of combining robust backend services with user-centric frontend designs to create applications that are both functional and intuitive. Future work could explore the incorporation of additional features, such as multimedia messaging and enhanced security protocols, to further enrich the user experience and maintain the application's relevance in the evolving digital landscape.

ACKNOWLEDGEMENT

The authors, Shraddha Varma, Sakshi Shinde and Swapnil Jadhav extend our sincere gratitude to our respective institutions for their unwavering support and the resources provided throughout this research. Special appreciation is directed to [Sakshi Shinde and Shraddha Varma] for their significant contributions to ["the development of the user interface"], and to [Swapnil Jadhav] for their dedicated work on ["the backend integration and database management"]. We are profoundly thankful to our Prof. Apeksha Pande, who is the project guide and coordinator of the Computer Engineering Department at Siddhant College of Engineering, for their invaluable guidance and continuous support during this project.

REFERENCES

1. Henriyan, D., Subiyanti, D. P., Fauzian, R., Anggraini, D., Aziz, M. V. G., & Prihatmanto, A. S. (2016). Design and Implementation of Web Based Real Time Chat Interfacing Server. *2016 6th International Conference on System Engineering and Technology (ICSET)*, 83–87.
2. Singh, S., Najam, S. S., & Sharma, A. (2023). Real-Time Secure Web-Based Chat Application using Django. *International Journal of Advances in Engineering and Management*, 5(2), 316–320.
3. Author, A. B., & Author, C. D. (2023). Real-Time Chat Application. *International Journal of Computer Applications*, 123(4), 45–50.
4. Author, E. F., Author, G. H., & Author, I. J. (2022). An Overview of Real-Time Chat Application. *International Journal of Research Trends and Innovation*, 7(6), 316–320.
5. Author, K. L., & Author, M. N. (2023). Web-Based Chat Application Using REACT. *SSRN Electronic Journal*.