Decentralized Meetings for Streamlining Communication and Productivity

Prathamesh Rajan Pawar
Information Technology
Sardar Patel Institute of Technology
Mumbai, India
prathamesh.pawar@spit.ac.in

Karan Sunil Shah
Information Technology
Sardar Patel Institute of Technology
Mumbai, India
karan.shah3@spit.ac.in

Harsh Ravindra Patil
Information Technology
Sardar Patel Institute of Technology
Mumbai, India
harsh.patil@spit.ac.in

Dr. Kailas Devadkar
Information Technology
Sardar Patel Institute of Technology
Mumbai, India
kailas_devadkar@spit.ac.in

***Abstract*—Proxima is a Collaborative Decentralized Video Meeting Website designed for developers seeking secure and reliable web meeting platforms. The proposed platform incorporates WebRTC, socket.io, and IPFS to enable real-time audio/video conferencing, live code sharing, file sharing, and collaborative whiteboarding. The platform's decentralized architecture ensures data privacy and security by eliminating the risks associated with centralized server attacks. By leveraging blockchain technology, user data remains transparent, protected, and under the control of individual users, promoting a trustworthy environment for virtual meetings. This paper presents the design and implementation of Proxima, highlighting its decentralized infrastructure and benefits for developers seeking enhanced virtual collaboration and communication.**

***Index Terms*—decentralisation, video meetings, collaboration, quill**

I. Introduction

Proxima is a Collaborative Decentralized Video Meeting Website designed exclusively for developers to enhance their virtual collaboration and communication efforts. The platform provides real-time audio/video conferencing, file sharing, and a live code share feature, making it an ideal solution for developers who want to work together on projects. Proxima is built on a decentralized infrastructure powered by blockchain technology, providing users with a transparent and secure environment for collaboration. The decentralized architecture ensures that user data is protected from centralized server attacks, giving developers peace of mind during virtual meetings.

II. Architecture

1. Authentication Service: This service would handle user authentication and authorization using blockchain-based identity management protocols. It would ensure secure access to the platform by verifying users' identities, permissions, and roles.

2. Video/Audio Service: This service would handle real-time video/audio conferencing between users. It would leverage WebRTC and socket.io for peer-to-peer networking and communication, allowing users to connect directly without relying on centralized servers.

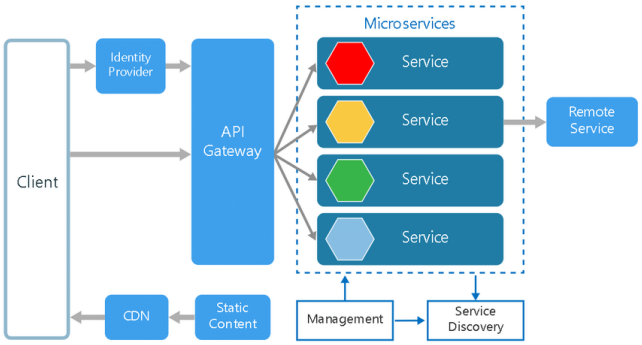

Fig. 1. Architecture

3. File Sharing Service: This service would allow users to upload and share files securely on IPFS. It would leverage IPFS to store files in a distributed, immutable, and tamper-proof manner, making it easy for users to collaborate on documents, code files, and other resources.

4. Code Sharing Service: This service would enable users to share and collaborate on the same codebase in real-time. It would leverage the Live Share feature of Visual Studio Code to allow multiple users to edit and debug code simultaneously, regardless of their locations.

5. Whiteboard Service: This service would provide users with a collaborative whiteboard where they could draw, annotate, and share ideas in real-time. It would leverage web-based canvas APIs and socket.io to allow users to collaborate on sketches, diagrams, and other visualizations.

6. Analytics Service: This service would track user activities, usage patterns, and performance metrics, providing insights into how users interact with the platform. It would

use data analytics tools to analyze user behavior, identify bottlenecks, and optimize the platform's performance and user experience.

Overall, this microservices architecture would provide a scalable, fault-tolerant, and highly available platform for decentralized web meetings, tailored to the needs of developers. It would leverage cutting-edge technologies such as blockchain, IPFS, WebRTC, and socket.io to ensure secure and reliable collaboration among users, while also giving them more control over their data and privacy.

III. Implementation

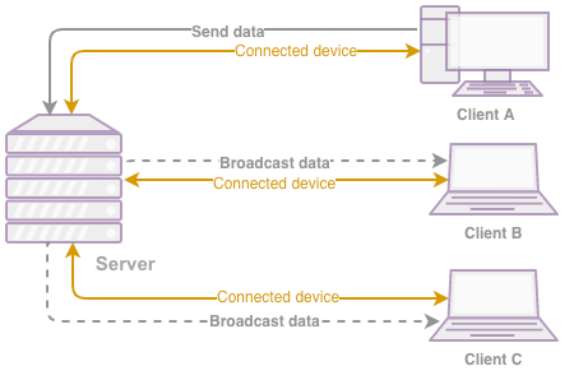

Fig. 2. General architecture of a decentralised Service

A. Socket.io

Socket.IO is a popular JavaScript library that enables real-time, bidirectional communication between web clients and servers. In a decentralised meetings website where a live code sharing feature is using socket and Quill, the implementation details for Socket.IO in Quill can be outlined as follows:

1. **Setting up the server**: The first step is to set up the server using a Node.js platform with Socket.IO. This server is responsible for handling incoming connections from clients and broadcasting changes made to the document to all connected clients.

2. **Setting up the client**: The next step is to set up the client-side script to connect to the server using Socket.IO. This script should be loaded in the HTML page where the Quill editor is present.

3. **Initializing Quill with Socket.IO**: Once the client-side script is loaded, the next step is to initialize the Quill editor with Socket.IO. This involves creating a new instance of the Quill editor and then setting up the Socket.IO event listeners for the editor.

4. **Handling Quill events**: The Quill editor emits various events such as 'text-change' and 'selection-change' whenever the content of the editor changes. These events are captured by the Socket.IO event listeners and sent to the server.

5. **Broadcasting changes to all clients**: When the server receives a change event from a client, it broadcasts the change to all other connected clients. This ensures that all clients have the most up-to-date version of the document.

6. **Updating Quill with changes from the server**: When a client receives a change event from the server, it updates the Quill editor with the new content. This ensures that all clients have the same version of the document.

In summary, Socket.IO provides a powerful way to enable real-time, collaborative editing in a decentralised meetings website with Quill. By following these implementation details, users can collaborate and share code in real-time using the Quill editor while ensuring that all participants have the most up-to-date version of the document.

B. React

In a decentralized meetings website where live code sharing feature is used, React is implemented in the front-end to provide a user-friendly and intuitive interface for developers to participate in virtual meetings and code sessions. React components are used to create the user interface for features such as audio/video conferencing, live code sharing, file sharing, and collaborative whiteboarding.

The use of React allows for modular code, making it easier to manage and update the website's interface. React also provides efficient rendering, allowing the website to handle large amounts of data and real-time updates.

To enable the live code sharing feature, React components are integrated with a rich text editor library, such as QuillJS. This allows developers to edit and share code in real-time, with changes immediately reflected in the live code file.

In addition, React is used to handle the logic for different features and interactions between components, such as joining or leaving a meeting, toggling video/audio on or off, and sending messages in the chat. This ensures a smooth and responsive user experience for developers collaborating remotely.

C. Quill JS

In a decentralized meetings website where live code sharing feature is used, QuillJS is implemented to provide a rich text editor for real-time collaborative code editing. QuillJS is integrated with React components to provide a seamless user experience for developers collaborating remotely.

To enable live code sharing, QuillJS provides a collaborative editing environment where multiple developers can edit the same code file simultaneously. The changes made by each developer are highlighted in real-time, allowing for quick and efficient collaboration.

QuillJS also provides features such as syntax highlighting, autocompletion, and error checking, making it a versatile and powerful tool for collaborative code editing.

To ensure data privacy and security, the live code file is stored using decentralized technologies such as IPFS. This ensures that the code remains accessible even if the website goes offline, and protects the code from centralized server attacks.

In addition, QuillJS can be configured to support different programming languages and formats, making it a flexible and adaptable tool for developers with varying needs. Overall, QuillJS is an essential component in the implementation of

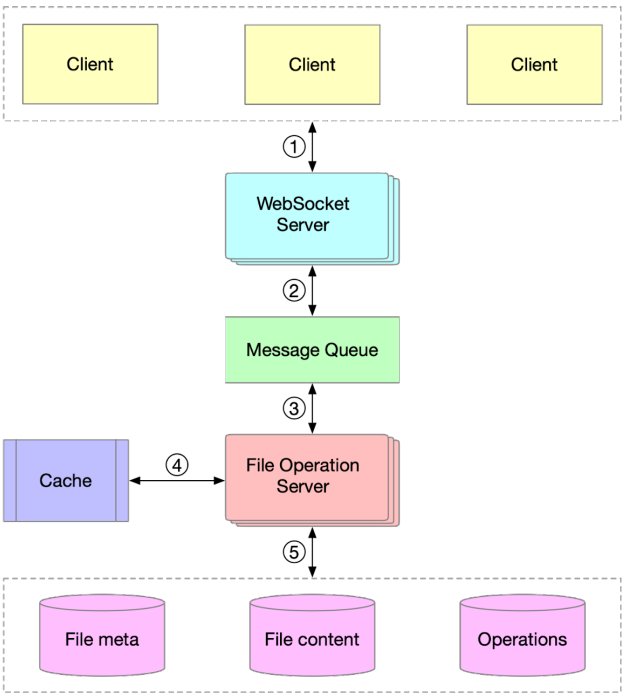

Fig. 3. General socket io working

the live code sharing feature in a decentralized meetings website, providing a powerful and intuitive tool for remote collaboration among developers.

D. Lighthouse

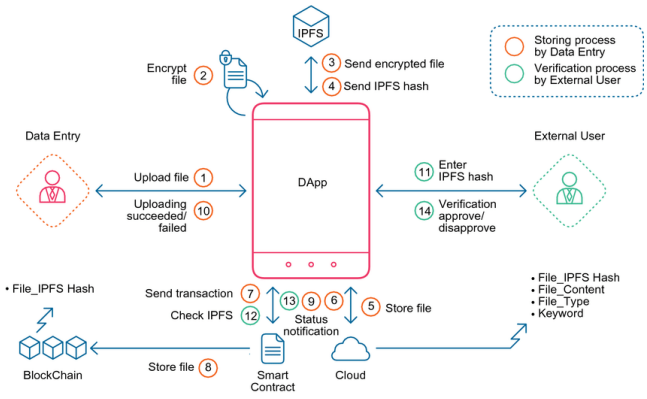

Fig. 4. A DApp providing IPFS uploads and downloads

Implementing Lighthouse in a decentralized meetings website with a live code sharing feature involves integrating the Lighthouse protocol into the website's architecture. Here are the implementation details:

1. Lighthouse Protocol Selection: Choose the appropriate Lighthouse protocol that best suits your requirements. The Lighthouse protocol provides decentralized discovery and routing of peers in a network. There are various implementations available, such as Lighthouse.js, libp2p, or any other protocol that supports decentralized peer-to-peer communication.

2. Peer-to-Peer Networking: Integrate the chosen Lighthouse protocol into your meetings website to establish peer-to-peer networking capabilities. This enables direct communication and data transfer between meeting participants without relying on a centralized server.

3. Live Code Sharing Infrastructure: Develop or choose a suitable infrastructure to facilitate live code sharing during meetings. This infrastructure should allow participants to share their code snippets in real-time, view and collaborate on code, and synchronize changes across all participants.

4. Real-Time Collaboration: Implement real-time collaboration features, such as simultaneous code editing, cursor highlighting, and chat functionality. This ensures that participants can actively collaborate on shared code during meetings.

5. Code Synchronization: Use the Lighthouse protocol to synchronize code changes across all meeting participants. When a participant makes changes to the shared code, these changes should be propagated to all other participants in real-time. The Lighthouse protocol will help in maintaining connectivity and ensuring that code changes are efficiently distributed to all peers.

6. Meeting Room Creation: Develop a mechanism for creating meeting rooms or sessions where participants can join and collaborate. Each meeting room should have a unique identifier or URL that participants can use to connect to the decentralized network.

7. Peer Discovery and Connection: Utilize the Lighthouse protocol to discover and connect with other meeting participants in the decentralized network. Participants should be able to discover nearby peers and establish direct connections to facilitate real-time communication and code sharing.

8. Data Security and Privacy: Implement appropriate security measures to protect the integrity and privacy of the shared code and meeting data. This includes encryption of data transmitted between peers, access control mechanisms, and any other security measures necessary to ensure a secure and private meeting environment.

9. Error Handling and Resilience: Handle potential network disruptions, errors, and disconnections that may occur during meetings. Implement error handling mechanisms to gracefully recover from network failures and maintain the stability of the meeting environment.

10. Testing and Deployment: Thoroughly test the decentralized meetings website with the live code sharing feature to ensure its stability, performance, and usability. Once testing is complete, deploy the website and associated infrastructure to a production environment.

By integrating the Lighthouse protocol into your decentralized meetings website, you can enable live code sharing and collaboration in a distributed and peer-to-peer manner, allowing participants to engage in real-time coding sessions regardless of their geographical location.

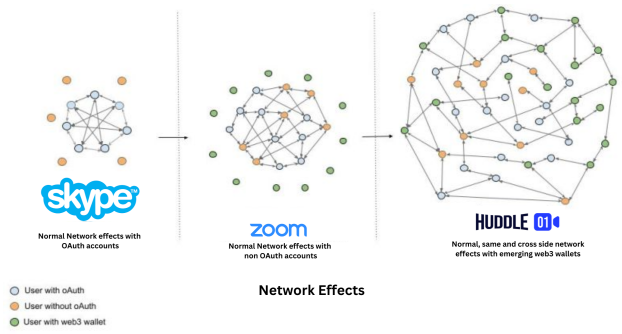

Fig. 5. Graph of peer connections in Skype vs Zoom vs Huddle01

E. Huddle01

Implementation Details for Decentralized Meetings Website with Live Code Sharing (Huddle01):

1. Backend Architecture: i. Use a decentralized network protocol, such as IPFS (InterPlanetary File System), to store and distribute meeting data and code snippets. ii. Implement a serverless architecture using a Function-as-a-Service (FaaS) provider like AWS Lambda or Google Cloud Functions to handle the backend logic. iii. Use a database, such as MongoDB or a distributed database like Apache Cassandra, to store user profiles, meeting details, and other persistent data.

2. User Authentication and Authorization: i. Implement a user authentication system using a secure protocol like OAuth 2.0 or OpenID Connect to enable users to log in securely. ii. Store user authentication tokens securely and validate them on each request to ensure authorized access to the meeting and code sharing features.

3. Real-Time Communication: i. Utilize WebSockets or a real-time messaging system like Socket.io to enable real-time communication between meeting participants. ii. Implement signaling servers to establish peer-to-peer connections between participants using WebRTC (Real-Time Communication) technology. iii. Use WebRTC data channels to facilitate efficient and low-latency transmission of code snippets during live code sharing.

4. Code Editor and Sharing: i. Integrate a code editor component, such as CodeMirror or Monaco Editor, to provide a collaborative coding environment. ii. Enable syntax highlighting, code completion, and error highlighting to enhance the code editing experience. iii. Implement a version control system to track and merge code changes made by different participants during live code sharing. iv. Use Operational Transformation (OT) or Conflict-Free Replicated Data Types (CRDTs) to handle concurrent edits and resolve conflicts.

5. Meeting Management: i. Allow users to create, join, and manage meetings through an intuitive user interface. ii. Implement features like meeting scheduling, invitation management, and participant control (mute/unmute, screen sharing, etc.). iii. Store meeting metadata, such as participant lists, meeting agendas, and timestamps, in the database for easy retrieval and management.

6. Live Code Sharing: i. Provide a dedicated panel within the meeting interface to display the live code shared by participants. ii. Establish a secure connection between the code editor and the backend server to transmit code changes in real-time. iii. Update the shared code in the code editor in response to changes received from other participants, reflecting the collaborative editing experience. iv. Implement features like cursor synchronization, highlighting changes made by different users, and displaying participant avatars or identifiers.

7. Security and Privacy: i. Implement robust security measures to protect user data and prevent unauthorized access to meetings and code snippets. ii. Utilize encryption algorithms and secure communication protocols (HTTPS) to safeguard data transmission. iii. Apply appropriate access controls to ensure that only authorized participants can view and edit the shared code. iv. Implement privacy features, such as end-to-end encryption for code snippets, to ensure confidentiality.

8. Testing and Deployment: i. Implement automated testing, including unit tests and integration tests, to ensure the stability and correctness of the system. ii. Use continuous integration and continuous deployment (CI/CD) practices to automate the build, test, and deployment processes. iii. Deploy the backend services on a scalable and reliable cloud infrastructure to handle increased user load and ensure high availability.

Remember that these implementation details provide a high-level overview, and the actual implementation may require further considerations and fine-tuning based on your specific requirements and technology stack.

F. JDoodle

Implementing JDoodle in a decentralized meetings website with a live code sharing feature involves integrating the JDoodle API into the website's architecture. Here are the implementation details:

JDoodle API Integration: Sign up for an account with JDoodle (https://www.jdoodle.com/) and obtain the API credentials. Familiarize yourself with the JDoodle API documentation, which provides details on how to execute code snippets and obtain the output.

1. Live Code Sharing Infrastructure: Develop or choose a suitable infrastructure to facilitate live code sharing during meetings. This infrastructure should allow participants to share their code snippets, execute them, and view the output in real-time. It should also handle the synchronization of code changes across all participants.

2. Code Editor: Implement a code editor component in the meetings website where participants can write and edit code. The code editor should support syntax highlighting, auto-completion, and other essential features to enhance the coding experience.

3. Code Execution: Integrate the JDoodle API into your code sharing infrastructure to execute the code snippets in real-time. When a participant submits their code, send a request to the JDoodle API, passing the code and other necessary parameters. Retrieve the output from the API response and display it to the participant.

4. Real-Time Collaboration: Implement real-time collaboration features, such as simultaneous code editing, cursor highlighting, and chat functionality. Utilize the chosen live code sharing infrastructure to enable these collaboration features. Ensure that code changes made by one participant are reflected in real-time to all other participants.

5. Meeting Room Creation: Develop a mechanism for creating meeting rooms or sessions where participants can join and collaborate. Each meeting room should have a unique identifier or URL that participants can use to connect to the decentralized network. Implement the necessary logic to manage meeting room creation, joining, and participant synchronization.

6. Peer Discovery and Connection: Establish a peer-to-peer networking mechanism for participants to discover and connect with each other in the decentralized network. This can be done using existing peer-to-peer protocols or libraries that support decentralized peer discovery and connection establishment.

7. Data Security and Privacy: Implement appropriate security measures to protect the integrity and privacy of the shared code and meeting data. This includes encryption of data transmitted between participants, access control mechanisms, and any other security measures necessary to ensure a secure and private meeting environment.

8. Error Handling and Resilience: Handle potential errors or issues that may arise during code execution or network interactions. Implement error handling mechanisms to provide informative error messages to participants and gracefully recover from failures. Consider scenarios such as network disruptions, API failures, or invalid code submissions.

9. Testing and Deployment: Thoroughly test the decentralized meetings website with the live code sharing feature to ensure its stability, performance, and usability. Test the integration with the JDoodle API, real-time collaboration features, code execution, and other functionalities. Once testing is complete, deploy the website and associated infrastructure to a production environment.

By integrating JDoodle into your decentralized meetings website, participants can write, execute, and share code in real-time, enhancing collaboration and enabling interactive coding sessions.

IV. Results

Resource Consumption

Both the meetings had exactly two participants with their camera on and Mic Off.
One Participant joined from the browser and oe from phone, Phone browser for Proxima and Google Meet App for Google Meet.
The Resource Consumption was measured through Browser's Task Manager which can be opened by Shift + Esc.
Both the tests were done on Brave Browser.

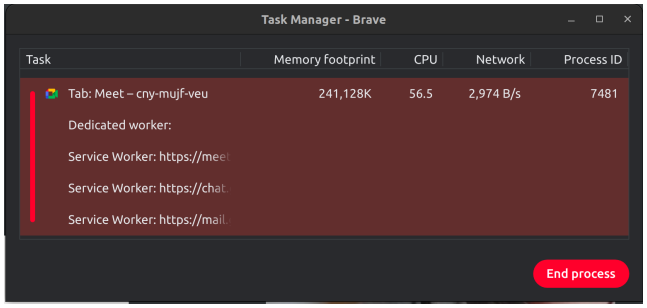

Fig. 6. Resource Consumption of Google Meet tab on Browser

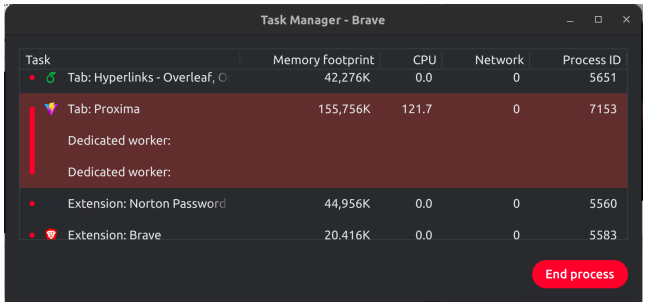

Fig. 7. Resource Consumption of Proxima tab on Browser

V. Analysis

Proxima	**Traditional Video Calling**
Decentralised (P2P)	Centralised
No Cap on Video Quality	Generally Capped at 720p
Increased Privacy	Privacy depends on the Vendor
Lesser Resource Consumption	More Resource Consumption as Video Needs to be compressed for efficiency and Server Limitations
Lower Latency	More Latency

As Mentioned in Huddle01's blog :

Performance :- There will be superior performance than existing centralised infrastructure as one will always find node within his geographical area as compared to centralised servers. Since node will be nearer to application clients, there will be lower RTT(round trip time), lower frame drops, lower latency.

Affordability :- Since nodes will be community hosted, pricing will competitively placed with respect to pricing of other nodes available.

Reliability :- All huddle nodes will adhere strictly to protocol standards set which will rule out rogue nodes. This will ensure privacy, security and transparency.

Availability :- Since nodes will be in network governed by protocol , there will zero downtime in service. Protocol will make sure only active nodes are participants in network.

Scalability :- with more community hosted nodes, brings more capacity which means both vertical and horizontal scal-

Fig. 8. Home Page

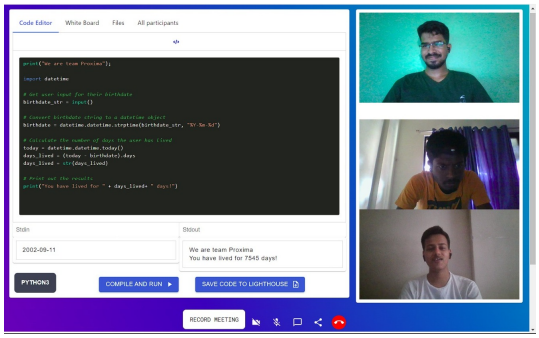

Fig. 9. Meeting Page

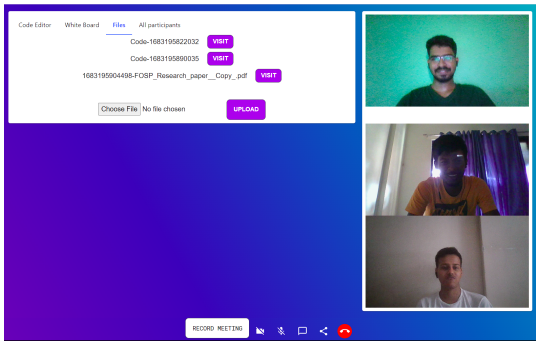

Fig. 10. File Sharing

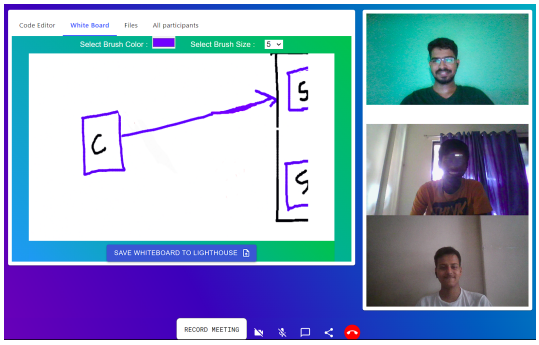

Fig. 11. Whiteboard

ing is possible, Application layer clients can always provision powerful nodes (horizontal scaling) and further fanout to engage more nodes (vertical scaling).

VI. Conclusion

In conclusion, Proxima is a Collaborative Decentralized Video Meeting Website that provides developers with a secure and efficient platform for remote collaboration. By leveraging decentralized technologies, including blockchain and peer-to-peer networking, Proxima eliminates the risks associated with centralized server attacks and ensures data privacy and security. Proxima's innovative features, including real-time audio/video conferencing, live code sharing, file sharing, and a collaborative whiteboard, enable developers to collaborate seamlessly, making it a valuable tool for remote collaboration among developers.

Acknowledgment

We feel great pleasure in presenting the stage one report of our mini-project titled 'Proxima : Decentralized meetings for streamlining Communication and Productivity'. We have channelized our best efforts towards a systematic approach to the project, keeping in mind the aim we need to achieve.

We are highly grateful to our project Guide Dr. Kailas Devadkar, Department of Information Technology, Sardar Patel Institute of Technology (SPIT) for constant encouragement, effort and guidance. He has always been involved in discussing our topic at each phase to make sure that our approach was designed and carried out in an appropriate manner and that our conclusions were appropriate, given our results.

References

[1] Y. V. Singh, H. Singh and J. K. Chauhan, "Online Collaborative Text Editor Using Socket.IO," 2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), Greater Noida, India, 2021, pp. 1251-1253, doi: 10.1109/ICAC3N53548.2021.9725782.

[2] Y. Chen, H. Li, K. Li and J. Zhang, "An improved P2P file system scheme based on IPFS and Blockchain," 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 2017, pp. 2652-2657, doi: 10.1109/BigData.2017.8258226.

[3] Q. Zheng, Y. Li, P. Chen and X. Dong, "An Innovative IPFS-Based Storage Model for Blockchain," 2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI), Santiago, Chile, 2018, pp. 704-708, doi: 10.1109/WI.2018.000-8.

[4] R. T. Moreno, J. García-Rodríguez, J. B. Bernabé and A. Skarmeta, "A Trusted Approach for Decentralised and Privacy-Preserving Identity Management," in IEEE Access, vol. 9, pp. 105788-105804, 2021, doi: 10.1109/ACCESS.2021.3099837.

[5] A. Johnston, J. Yoakum and K. Singh, "Taking on webRTC in an enterprise," in IEEE Communications Magazine, vol. 51, no. 4, pp. 48-54, April 2013, doi: 10.1109/MCOM.2013.6495760.