International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Empowering Collaborative Programming: The Colab Code Strategy for Consistency and Awareness

[1]Mr. Girish Navale, [2]Dr. Pallavi V Baviskar, [3]Ms.Shital Abhimanyu Patil, [4]Indira P. Joshi, [5]Dr. Shraddha R. Khonde , [6]Miss. Sneha Ramdas Shegar

Submitted: 04/02/2024 **Revised**: 12/03/2024 **Accepted**: 18/03/2024

Abstract: The ColabCode is a ground-breaking solution that empowers geographically dispersed individuals working on the same project to collaborate seamlessly and simultaneously. Leveraging advanced technologies and an intuitive user interface, ColabCode provides a comprehensive platform where participants can collaborate, share code, and communicate effectively in real time.ColabCode addresses the challenges faced by distributed teams, allowing them to work cohesively on shared projects. By providing a unified workspace accessible from anywhere, participants can log in to the platform and instantly join the project in progress. Regardless of their location or time zone, team members can start working together, eliminating delays caused by coordination issues. The heart of ColabCode lies in its diverse programming language support. With built-in capabilities for various programming languages such as Python, Java, JavaScript, C++, and more, team members can seamlessly contribute code in their preferred language. This flexibility fosters inclusivity and encourages diverse skill sets, ensuring each member can work efficiently using their expertise.

Keywords*: real-time collaborative editor, remote collaboration, code sharing, code synchronization*

1. Introduction

The advent of remote work has transformed the way teams collaborate and contribute to projects. However, when working on the same project from different locations,[1] teams often face challenges such as coordination issues, communication gaps, and difficulties in sharing and synchronizing code. To address these obstacles, we present the ColabCode, a powerful platform that revolutionizes remote collaboration by enabling simultaneous work and code sharing across multiple programming languages[2-4].A key feature of ColabCode is its support for multiple programming languages. Recognizing that different team members may have expertise in diverse languages, ColabCode provides an environment that caters to individual preferences. Whether one prefers Python, Java, JavaScript, C++, or any other language, team members can contribute code seamlessly, fostering an inclusive environment that harnesses the strengths and talents of each individual[5-10].In addition to code sharing, ColabCode incorporates a comprehensive chat feature, enabling real-time communication among team members. The chat functionality allows for instant discussions, brainstorming sessions, and problem-solving, fostering effective collaboration even when physically separated. Participants can exchange ideas, seek feedback, and provide support, creating a cohesive and interactive virtual workspace[11-13].

2. Literature Survey

In today's increasingly interconnected world, remote work has become a prevalent practice, enabling teams to collaborate regardless of their physical locations. However, remote collaboration brings its own set of challenges, particularly when it comes to working on shared projects and sharing code across multiple programming languages. The ColabCode addresses these challenges by providing a unified platform that enables simultaneous remote collaboration and code sharing across various programming languages[12].The domain of the ColabCode lies at the intersection of remote work, collaborative software development, and project management. It recognizes the need for remote teams to have a centralized workspace where they can work together seamlessly, communicate effectively, and contribute code in their preferred languages.When developing the ColabCode[14-15], comprehensive comparison research was conducted to evaluate existing tools and platforms that enable remote collaboration and

[1]Assistant Professor, Computer Engineering department, Aissms Institute Of Information Technology Pune
[2]Assistant Professor, Computer Engineering department, Sandip Institute of Engineering and Management, Sandip Foundation Nashik
[3]Assistant Professor, Computer Engineering department, SSBT COET Bambhori
[4]Assistant Professor, Computer Engineering department, NHITM clg Thane West.
Maharashtra. Mumbai University.
[5]Assistant Professor, Computer Engineering department, Modern Education Society's Wadia College of Engineering Pune
[6]HOD, Computer Engineering department, Samarth Group of Institutions College of Engineering, Belhe
Email - girish.navale@aissmsioit.org, palbaviskar@gmail.com ,shital.a.patil2014@gmail.com, ipj.indira@gmail.com, khonde.shraddha@gmail.com, snehashegar1@gmail.com

code sharing. The purpose of this research was to identify the strengths, weaknesses, and unique features of these tools, in order to ensure that CPH provides a distinct and valuable solution for remote teams.By conducting this comprehensive comparison research, ColabCode identified the gaps and opportunities in existing tools and platforms. It enabled the development team to understand the requirements of remote teams, anticipate potential challenges, and incorporate unique features and functionalities into ColabCode. The research findings ensured that ColabCode offers a distinct and valuable solution[16-19], addressing the limitations of existing tools and providing an enhanced collaborative experience for remote teams working on shared projects and code sharing across multiple programming languages.

1. GitHub: GitHub is a widely used platform for version control and code collaboration. It provides features such as code repositories, issue tracking, pull requests, and project management. While it supports collaboration, it does not offer simultaneous real-time collaboration or built-in support for multiple programming languages.

2. GitLab: GitLab is another popular platform that offers version control, code collaboration, and CI/CD (continuous integration and continuous deployment) capabilities. It provides features similar to GitHub, including code review, issue tracking, and project management. GitLab also offers real-time editing for certain file types, but it may not support simultaneous collaboration across multiple programming languages.

3. Visual Studio Live Share: Visual Studio Live Share is an extension for Visual Studio Code that enables real-time collaboration on code editing, debugging, and running applications. It allows developers to share their development environment and work together on the same codebase in real-time. However, it may have limitations in terms of supporting multiple programming languages.

4. JetBrains Space: JetBrains Space is an integrated team collaboration platform that offers a range of tools for software development teams. It provides features such as code hosting, version control, project

management, and chat. JetBrains Space aims to provide a comprehensive platform for remote collaboration, but its specific support for simultaneous collaboration and multiple programming languages may vary.

1. Visual Studio Code (VSCode)

2. Eclipse

3. IntelliJ IDEA

4. PyCharm

5. Sublime Text

6. Atom

7. Xcode (for macOS and iOS development)

8. NetBeans

9. Android Studio (for Android app development)

10. Jupyter Notebook

11. MATLAB

12. RStudio (for R programming)

13. Arduino IDE (for Arduino development)

14. Unity (for game development)

15. Scratch (for educational programming)

These platforms offer different features, languages, and target specific domains. They range from general-purpose code editors to integrated development environments (IDEs) tailored for specific languages or frameworks. It's important to choose the platform that aligns with our programming needs and preferences. Many of the programming platforms mentioned, programmers have the ability to work in parallel or collaborate with others. However, the extent of parallel work and collaboration capabilities may vary depending on the specific platform. Here are a few examples:

1. Visual Studio Code (VSCode): While VSCode is primarily designed for individual development, it offers extensions like Live Share that enable real-time collaboration, allowing multiple developers to work on the same codebase simultaneously.

2. Eclipse: Eclipse supports team collaboration through features like collaborative editing, version control integration (such as Git), and project sharing capabilities. Developers can work together on shared projects and access common resources.

3. IntelliJ IDEA and PyCharm: These IDEs provide built-in support for collaboration and version control systems. Developers can use features like code sharing, code reviews, and remote development, enabling parallel work on projects.

4. Jupyter NotebookJupyter Notebook is a popular platform for interactive computing and data analysis. It supports collaboration by allowing multiple users to work on the same notebook concurrently, with each user able to see the changes made by others in real-time.

5. Google Colab: As a cloud-based platform, Google Colab is designed for collaboration. Multiple users can simultaneously work on the same Colab notebook, making it suitable for group projects or collaborative coding sessions.

6. GitHub/Bitbucket/GitLab: While not programming platforms in themselves, these version control platforms provide collaboration features such as code reviews, issue tracking, and pull requests. They facilitate parallel work and coordination among developers working on the same codebase.

3. Drawbacks of Existing Platforms-

While the programming platforms mentioned in the previous response offer numerous benefits, they also have some potential drawbacks. Here are some common drawbacks associated with these platforms:

1. Learning Curve: Some platforms, particularly full-featured IDEs like Eclipse, IntelliJ IDEA, and PyCharm, can have a steep learning curve for beginners. Their extensive features and configurations may require some time and effort to become proficient in using them effectively.

2. Resource Requirements: Certain platforms, especially those with rich features and extensive tooling support, can be resource-intensive. They may require a significant amount of memory and processing power, potentially affecting the performance of older or less powerful machines.

3. Complexity: With increased functionality and flexibility, programming platforms can sometimes become complex. This complexity may be overwhelming for novice programmers or those seeking simplicity in their development environment.

4. Customization Overload: Some platforms, like Visual Studio Code, offer a vast number of extensions and customization options. While this can be advantageous for tailoring the platform to your specific needs, it can also lead to decision fatigue or spending excessive time searching for and configuring the right set of extensions.

5. Limited Language Support: Certain platforms may be designed with a primary focus on specific programming languages or frameworks, potentially limiting their effectiveness for projects involving multiple languages or less commonly used languages.

6. Offline Limitations: Cloud-based platforms, such as Google Colab, require an internet connection to access and run code. This can be a disadvantage if you need to work in offline environments or have limited connectivity.

7. Lack of Integration: Some platforms may have limited or no built-in integration with certain tools or technologies, requiring developers to find and configure additional plugins or workarounds to fulfill their specific requirements.

8. Platform Lock-In: Some platforms have proprietary formats or workflows that may make it difficult to migrate projects to other platforms or IDEs, potentially leading to vendor lock-in.

It's important to note that these drawbacks may vary in significance depending on individual preferences, project requirements, and the specific programming tasks at hand. Ultimately, it's essential to evaluate these factors and choose a platform that best aligns with your needs and provides the right balance between functionality and usability.

4. Algorithm

Yjs is a JavaScript framework that enables collaborative editing and real-time synchronization of shared data structures among multiple users. When it comes to integrating CodeMirror with Yjs, the goal is to create a collaborative code editor where changes made by one user are instantly reflected for all other connected users. Yjs achieves this by using a data type called a Shared Type, specifically the Y.Text type, which represents a shared text document.

Algorithmic explanation of how to achieve real-time synchronization using Yjs:

1. Initialize the shared document: Create a new instance of Y.Doc to represent the shared document. Add the required shared types (e.g., Y.Text) to the document to represent different data structures.
2. Set up communication: Establish a connection between clients using a communication channel like Web Socket. Implement the necessary event handlers or listeners to send and receive messages between clients.
3. Share the document: Once the connection is established, share the Y.Doc instance with other clients in the session. Transmit the shared document to connected clients using the communication channel.
4. Handle local changes: Listen for local changes made by the user in the editor or input fields. When a change occurs, update the corresponding shared type (e.g., Y.Text) in the Y.Doc instance.
5. Handle remote changes: Implement event handlers or listeners to receive incoming messages from other clients. Extract the changes made by remote clients, including the affected shared type and the corresponding delta. Apply the received changes to the corresponding shared type in the local Y.Doc instance.

6. Synchronize changes: When local changes are made, transmit the delta to other clients via the communication channel. Receive incoming deltas from other clients and apply them to the local Y.Doc instance.
7. Update the user interface: Monitor changes in the shared types within the Y.Doc instance. When changes occur, update the user interface elements (e.g., CodeMirror editor) to reflect the modifications.
8. Terminate the session: Implement a termination mechanism to gracefully close the connection and end the collaboration session. Perform any necessary cleanup tasks, such as removing event listeners and releasing resources.

The step-by-step algorithm explaining how to bind CodeMirror with Yjs without diving into code details:

1. Set up CodeMirror: Create an instance of CodeMirror and attach it to an HTML element on your web page. Specify configuration options such as the theme, language mode, and initial content.
2. Create a Yjs document: Initialize a new instance of Y.Doc, which represents the shared document. Attach a Y.Text type to the document. Y.Text is used to represent shared text data.
3. Bind CodeMirror with Yjs: Listen for the 'change' event in CodeMirror, capturing changes made by the local user.Retrieve the changes made (delta) and apply them to the Y.Text shared type.
4. Observe changes from Yjs: Set up an observer on the Y.Text type to detect changes made by other users. When a change is detected, retrieve the delta (changes) and apply them to CodeMirror.
5. Connect to a shared editing session: Establish a connection to a shared editing session using a communication channel like WebSocket. Use the appropriate Yjs provider (e.g., WebsocketProvider) to exchange updates between users.Share the Y.Doc instance in the editing session to enable real-time collaboration.

Yjs is a JavaScript framework that enables collaborative editing and real-time synchronization of shared data structures among multiple users. When it comes to integrating CodeMirror with Yjs, the goal is to create a collaborative code editor where changes made by one user are instantly reflected for all other connected users. Yjs achieves this by using a data type called a Shared Type, specifically the Y.Text type, which represents a shared text document.

Real-Time synchronization using Yjs

Create a new instance of Y.Doc to represent the shared document. Add the required shared types (e.g., Y.Text) to the document to represent different data structures. Establish a connection between clients using a communication channel like WebSocket. Implement the necessary event handlers or listeners to send and receive messages between clients. Share the document: Once the connection is established, share the Y.Doc instance with other clients in the session. Transmit the shared document to connected clients using the communication channel. Listen for local changes made by the user in the editor or input fields. When a change occurs, update the corresponding shared type (e.g., Y.Text) in the Y.Doc instance. Implement event handlers or listeners to receive incoming messages from other clients. Extract the changes made by remote clients, including the affected shared type and the corresponding delta. Apply the received changes to the corresponding shared type in the local Y.Doc instance. When local changes are made, transmit the delta to other clients via the communication channel. Receive incoming deltas from other clients and apply them to the local Y.Doc instance.

Update the user interface: Monitor changes in the shared types within the Y.Doc instance. When changes occur, update the user interface elements (e.g., CodeMirror editor) to reflect the modifications. Implement a termination mechanism to gracefully close the connection and end the collaboration session. Perform any necessary cleanup tasks, such as removing event listeners and releasing resources.

Codemirror binding with Yjs

Create an instance of CodeMirror and attach it to an HTML element on your web page. Specify configuration options such as the theme, language mode, and initial content. Initialize a new instance of Y.Doc, which represents the shared document. Attach a Y.Text type to the document. Y.Text is used to represent shared text data. Listen for the 'change' event in CodeMirror, capturing changes made by the local user. Retrieve the changes made (delta) and apply them to the Y.Text shared type. Set up an observer on the Y.Text type to detect changes made by other users. When a change is detected, retrieve the delta (changes) and apply them to CodeMirror. Establish a connection to a shared editing session using a communication channel like WebSocket. Use the appropriate Yjs provider (e.g., WebsocketProvider) to exchange updates between users.

Share the Y.Doc instance in the editing session to enable real-time collaboration.

5. Prototype Implementation and Evaluations

Following the system architecture in figure 2 , the workflow and functional design in Section figure 1(a),(b) and various techniques proposed and devised in Section 5, we have successfully implemented the CoVSCode prototype. In this section, we demonstrate the prototype system and present preliminary user evaluations along with a comprehensive set of experimental evaluations.

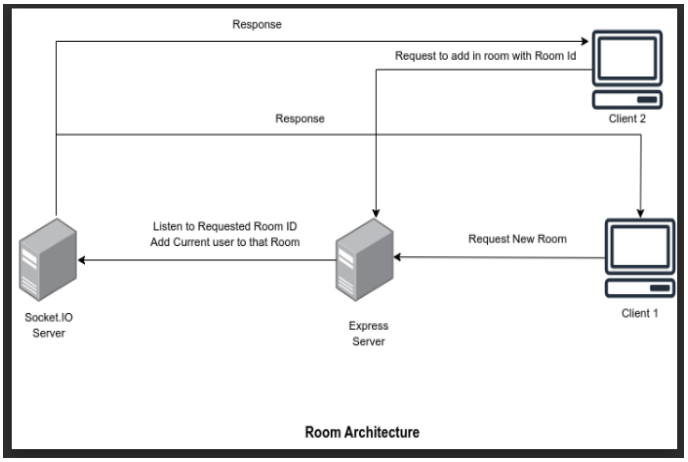

Fig 1(a)- Proposed Model Flow

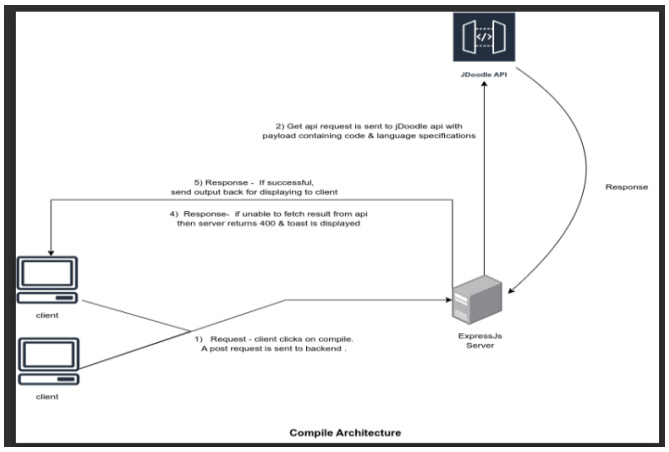

Fig 1(b)- Proposed Model Flow

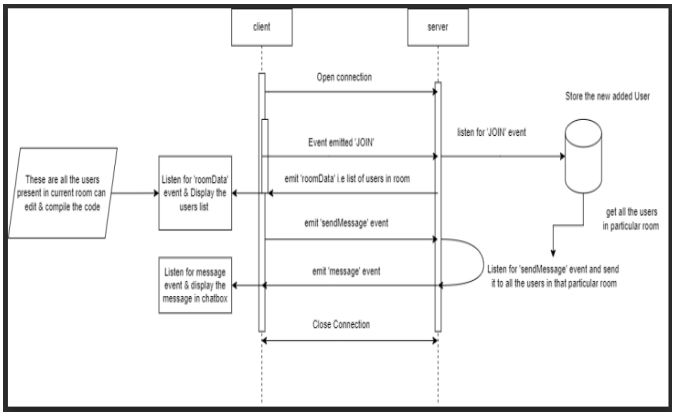

Fig 2- Proposed Architectural Diagram

6.1 Major User Interfaces of proposed Prototype System

6.1.1 Login and Initialization: Preparation of Real-Time Collaboration

Figure 3 presents the proposed client UI for a programmer to start real-time collaborative programming. Upon adding room ID , a login panel is displayed in the IDE. The programmer don't know have room ID then simply create new room.

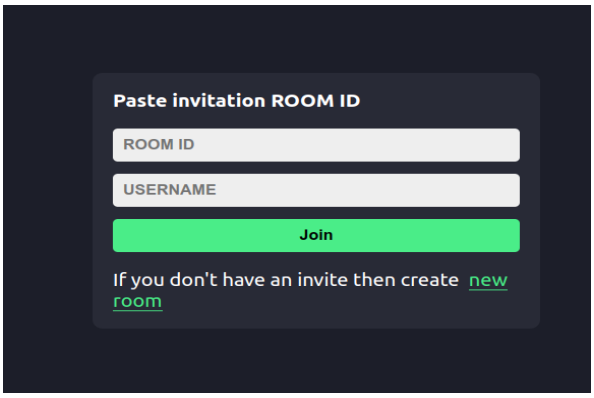

Fig 3- UI snapshot of the proposed client's login panel.

Client-side -

As you can see in the image below First we need to listen to the endpoint/port on which the server is running and need to emit(i.e. send) reserved event 'JOIN' which is used for making connections with the server. We are passing roomId and username as a payload.

```
// send the user to server so that we can add this user in room
useEffect(() => {
    socket = io(ENDPOINT)
    const username = location.state.username
    // console.log("sending roomId and user name to server")
    socket.emit("JOIN", {
        roomId,
        username
    }, (error) => {

        if (error) alert(error)
    });
    return () => {
        // socket.disconnect()
        socket.off("JOIN");
    }
}, []);
```

Fig 4 -UI snapshot of the CoVSCode client's initialization panel.

On the server side -

As we can see in the image below, first we need to use on('connection') which establishes the initial connection and sets up event listeners for that particular client. After that we listen for the event 'JOIN' which we have emitted from the client side.

```
io.on('connection', (socket) => {
    console.log('Socket Connected', socket.id)

    socket.on("JOIN", ({ roomId, username }, callback) => {

        console.log(roomId)
        console.log(username)
        const { error, user } = addUser({ id: socket.id, username, roomId })

        if (error) return callback(error)

        // userSocketMap[socket.id]= username;
        console.log("inside server-", (user))
        socket.join(user.roomId);

        console.log("Inside server", user.username)
        // users = getUsersInRoom(user.roomId)
        let kname = user.username;
        socket.emit('message', { user: 'admin', text: `${user.username}, welcome to room!` });
        socket.broadcast.to(user.roomId).emit('message', { user: 'admin', text: `${user.username} has joined!` });
```

Fig 5- On Server side

In short, io. on is responsible for the overall connection from client to server and then socket.on / emit will be responsible for specific clients' sockets.

Further, as we can see in the image we have added the user temporarily in an ArrayList and emitting and message using the 'message' event which is from the server side. This event is further listened to on the client side and will be available for newly joined users in the chat box.

The **socket.broadcast.to** is used for broadcasting the message i.e. all the users in that particular room will get the notification of this user being joined except the new user. This is one of the good features from the socket.io library.

Server side –

```
io.to(user.roomId).emit('roomData', { roomId: user.roomId, users: getUsersInRoom(user.roomId) })
```

Client side for sending messaging -

As you can see in the image below, We are sending the message which we have stored in a usestate message using an event named 'sendMessage' which will listen on the server.

```
//function for sedning message
const sendMessage = (event) => {
    event.preventDefault();
    if (message) {
        socket.emit('sendMessage', message, () => {
            setMessage("")
        });
    }
}
```

SendMessage on server side -

Here we are listening to the event 'message' which we have emitted from the client side. First we will get user details from the socket.id then we will send the message to all the users in that particular room using io.to(roomId). This will be done using the same emit method which we will listen on the client side.

```
socket.on('sendMessage', (message, callback) => {
    const user = getUser(socket.id);
    // console.log("Inside sendMessage", user)
    io.to(user.roomId).emit('message', { user: user.username, text: message });
    callback();
});
```

SendMessage listening on client side -

Here in client code, we are listening for that emit 'message' from the server for all the users. As we have seen that emit took place using io not socket so that was for all the users and not for specific client's socket. Here we listen for that message and populate our message useState which will be used for displaying in the UI.

```
// hook for receiving msg
useEffect(() => {
    socket.on('message', message => {
        console.log("client-side", message)
        setMessages(messages => [...messages, message]);

    });

    return () => {
        socket.off('message')
    }

}, [messages]);
```

So the above explanation is how we are storing users in a particular room and how they are able to chat with each other while working on code using socket.io.

6. Conclusion

Real-time code editors have made significant strides in revolutionizing collaborative coding. With their ability to enable simultaneous editing, live synchronization, and seamless communication, these editors have transformed the way developers collaborate and work together on code. Real-time code editors have successfully implemented features such as real-time collaboration, allowing multiple users to edit the same code simultaneously, compile and see changes in real-time. This functionality has greatly improved productivity and teamwork among developers, enabling them to work together efficiently and provide instant feedback.

CodeMirror, a popular code editor library, has been integrated with real-time synchronization frameworks like Yjs and communication technologies like Socket. IO. This integration allows for seamless communication between the editor and the server, ensuring that changes made by one user are immediately synchronized and reflected in the editor for all connected users.

Conflicts of Interest

The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Reference:

[1] M. Ahmed-Nacer, C.-L. Ignat, G. Oster, H.-G. Roh, and P. Urso. Evaluating CRDTs for Real-time Document Editing. In Proceedings of the 11th ACM symposium on Document Engineering, pages 103–112. ACM New York, 2011

[2] G. Oster, P. Urso, P. Molli, and A. Imine. Data Consistency for P2P Collaborative Editing. In CSCW ’06, pages 259–268. ACM Press.

[3] DESIGN & IMPLEMENTATION OF A REAL-TIME CHAT APPLICATION Al-Riyami, SS and K.G. Paterson, 2003. Uncertified public key cryptography. Methods for the Ninth World Theoretical Conference furthermore, Use of Cryptology and Information Security, November 30- Dec. 4, Springer Berlin Heidelberg, Taiwan, pages: 452-473. DOI: 10.1007/978-3-540-40061-5_29 Azab, A., P. Watters and R. Layton, 2012

[4] A. R. S. Gerlicher. A Framework for Real-time Collaborative Engineering in the Automotive Industries. In CDVE’06, pages 164–173. Springer-Verlag, 2006.

[5] S. Kumawat, M. T. Scholar, and A. Khunteta, “A Survey on Operational Transformation Algorithms: Challenges, Issues and Achievements", International Journal of Engineering Science and Technology, Vol. 2, No. 7, (2010), pp. 3311–3319.

[6] M. Goldman, G. Little, and R. C. Miller, “Real-time Collaborative Coding in a Web IDE”, in Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, pp. 155-164.

[7] A. Sarma, “A Survey of Collaborative Tools in Software Development, Technical Report, UCI-ISR-05-3”, Irvine, California

[8] CodeR: Real-time Code Editor Application for Collaborative Programming December 2015 Procedia Computer Science 59:510-519

[9] International Research Journal of Engineering and Technology (IRJET), BROWSER BASED CODE EDITOR

[10] https://www.irjet.net/archives/V8/i5/IRJET-V8I5440.pdf

[11] A Review on Server-Based Code Editor Sonali R. Gujarkar, Samprada D.Nimrad, Shital Meshram

[12] INTERNATIONAL JOURNAL FOR RESEARCH IN EMERGING SCIENCE AND TECHNOLOGY, SPECIAL ISSUE-1-JAN-2017

[13] W. Kimpan, T. Meebunrot, and B. Sricharoen, "Online code editor on Private cloud computing," 2013 International

[14] Computer Science and Engineering Conference (ICSEC), Nakhonpathom, Thailand, 2013, pp. 31-36, doi:10.1109/ICSEC.2013.6694748

[15] H.-G. Roh, M. Jeon, J.-S. Kim, and J. Lee. Replicated Abstract Data Types: Building Blocks for Collaborative Applications. Journal of Parallel and Distributed Computing, 71(3):354–368, 2011.

[16] Wankhede, D.S., Pandit, S., Metangale, N., Patre, R., Kulkarni, S., Minaj, K.A. (2022). Survey on Analyzing Tongue Images to Predict the Organ Affected. In: Abraham, A., et al. Hybrid Intelligent Systems. HIS 2021. Lecture Notes in Networks and Systems, vol 420. Springer, Cham. https://doi.org/10.1007/978-3-030-96305-7_56

[17] D. Wankhede, V. Mishra, M. Karnik, A. Kekane and A. Shukla, "The Impact of the Latest Technology on Healthcare and how can it be leveraged to improve patient outcomes and reduce Healthcare costs," 2023 4th IEEE Global Conference for Advancement in Technology (GCAT), Bangalore, India, 2023, pp. 1-6, doi: 10.1109/GCAT59970.2023.10353516.

[18] Sutar, S., Jose, K., Gaikwad, V., Mishra, V., Wankhede, D., & Karnik, M. (2023). Enhancing Data Management: An Integrated Solution for Database Backup, Recovery, Conversion, and Encryption Capabilities. International Journal of Intelligent Systems and Applications in Engineering, 12(6s), 720–734. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/4011

[19] Sakhare, N. N., Bangare, J. L., Purandare, R. G., Wankhede, D. S., & Dehankar, P. (2024). Phishing Website Detection Using Advanced Machine Learning Techniques. International Journal of Intelligent Systems and Applications in Engineering, 12(12s), 329 –. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/4519

[20] P. M, D. S. Wankhede, R. Kumar, G. Ezhilarasan, S. Khurana and G. S. Sahoo, "Leveraging AI-Driven Systems to Advance Data Science Automation," 2023 International Conference on Emerging Research in Computational Science (ICERCS*)*, Coimbatore, India, 2023, pp. 1-7, doi: 10.1109/ICERCS57948.2023.10434009.

[21] Mrs. Disha Sushant Wankhede, Dr. Selvarani Rangasamy,"REVIEW ON DEEP LEARNING APPROACH FOR BRAIN TUMOR GLIOMA ANALYSIS" Journal of Information Technology in Industry, VOL. 9 NO. 1 (2021) pp. 395 - 408 , DOI: https://doi.org/10.17762/itii.v9i1.144

[22] D. Rawat, A. Chaubey, P. Kumar, S. HEMELATHA, P. GAJENDRAN and D. S. Wankhede, "Examining 6G Infrastructure Capabilities: Paving the Way for Future Connectivity," *2023 International Conference on Power Energy, Environment & Intelligent Control (PEEIC)*, Greater Noida, India, 2023, pp. 594-597, doi: 10.1109/PEEIC59336.2023.10451019.