# Live Coder : For Collaborative Coding Sessions

**A Project Report**

*Submitted by*


| | |
|---|---|
| **Avanti More** | **112103089** |
| **Renuka Padalkar** | **112103100** |


*of*

## TY (Computer Engineering)

Under the guidance of

**Prof. Jibi Abraham**

COEP Technological University



**DEPARTMENT OF COMPUTER ENGINEERING**

**COEP Technological University**


April, 2024

## DEPARTMENT OF COMPUTER ENGINEERING

## COEP Technological University

# CERTIFICATE

Certified that this project, titled "Live Coder : For Collaborative Coding Sessions" A Project Report Submitted by Avanti More 112103089 Renuka Padalkar 112103100 of TY (Computer Engineering) Under the guidance of Prof. Jibi Abraham COEP Technological University DEPARTMENT OF COMPUTER ENGINEERING COEP Technological University April, 2024 " has been successfully completed by

| | |
|---|---|
| **Avanti More** | **112103089** |
| **Renuka Padalkar** | **112103100** |

and is approved for the fulfilment of the requirements of "Software Engineering Mini Project- Stage II".

SIGNATURE

**Prof. Jibi Abraham**

**Project Guide**

**Department of Computer Engineering**

**COEP Technological University,**

**Shivajinagar, Pune - 5.**

**Abstract**

The Live- Coder project aims to develop a comprehensive web application that empowers users to check,brainstorm and debug codes together. In a dynamic and competitive environment in the job market, coders often opt pair-programming and want to take mock interviews of each other. The Live Coder addresses this need by leveraging collaborative editor techniques and presenting it through a user-friendly interface.

The project methodology involves thorough requirements analysis, technology selection, system design, implementation, and testing. Key features of Live Coder include real-time collaborative code sharing in multiple programming C-like languages with integrated video calling ,messaging capabilities ensuring seamless collaboration and enhanced productivity for developers. The development process adheres to best practices in software engineering ensuring scalability, security, and usability.

# Contents

# List of Figures

# Chapter 1

# Synopsis

## 1.1 Project Title

Live Coder : For Collaborative Coding Sessions

## 1.2 Internal Guide

Prof. Jibi Abraham

## 1.3 Problem Statement

There is a pressing need for a comprehensive collaborative code editing platform that gives seamless communication features to help coders prepare for their real time interviews .Existing tools often lack the necessary features on a single platform to support this crucial aspect of coding preparation, resulting in fragmented communication and limited collaboration opportunities.

# Chapter 2

# Problem Definition and scope

## 2.1 Problem Definition

### 2.1.1 Goals and objectives

Goal and Objectives:

- Goals
  - Create a platform that helps coders practice for real-time interviews by enabling seamless communication and collaboration through features like integrated video calling.
- Objectives
  - **Real-time coding interface :**Allows users to engage in collaborative coding sessions. This feature aims to enhance communication, collaboration, and problem-solving by enabling users to discuss code changes, share insights, and provide immediate feedback during coding sessions.
  - **Invite users :**Allows one user (Host) to seamlessly invite another user (Guest) to participate in real-time coding sessions.

- **Communication capabilities:**Integrate robust video and audio communication.
- **Chatting interface :**Implement real-time messaging and chat functionality alongside the coding interface.
- **Syntax highlighting :**To incorporate syntax highlighting functionality that dynamically adjusts based on the programming language being used within the coding interface.

### 2.1.2 Statement of scope

- **Target Audience:** This application is ideal for recruiters who are looking to take online interviews. It will be particularly beneficial for individuals who want to prepare for these interviews.

## 2.2 Software context

The Collaborative Code Editor and Video Calling Web Application project amalgamates advanced technologies to provide developers with a dynamic platform for real-time collaboration. Utilizing HTML, CSS, and JavaScript, the interface is designed to be intuitive and visually appealing. Node.js, with Express.js, forms the robust backend, ensuring smooth server-side operations.

Socket.io enables seamless bidirectional communication, facilitating collaborative coding sessions where multiple users can work simultaneously. Peerjs adds video calling capabilities through peer-to-peer connections. Uuid ensures secure user identification. CodeMirror offers a versatile code editor, empowering developers with essential features. Together, these technologies create an immersive environment for distributed teams, fostering seamless collaboration and innovation in software

development.

## 2.3 Major Constraints

- **Data acquisition**
  - **Website structure and its changes:** Adapting to variations in website layouts and data formats.
  - **Security:** The techniques must comply with website terms of service and keep in mind data privacy regulations.
- **Client-side constraints**
  - **Synchronization of Data:** We need to ensure the consistency of video and changes in the editor on client-side in case of many users.

## 2.4 Outcome

The primary outcome of the project is the successful development and deployment of a comprehensive web application that empowers users to code collaboratively. Key components of the outcome include:

- **User-Friendly Interface :** A user-friendly interface that enables users to easily use the application, code efficiently, and communicate through video and audio.
- **Real-Time :** The application provides users with up-to-date changes.
- **Personalized sessions:** The ability for users to set up personalized sessions with no collision of same session code.
- **Enhanced User Experience:** The outcome results in an enhanced user experience, enabling users to make interactive sessions better.

## 2.5 Applications

- **Pair Programming:** Real-time coding collaboration platforms facilitate pair programming sessions where two developers work together on the same codebase simultaneously, enhancing code quality and knowledge sharing.
- **Code Reviews:** Teams can conduct real-time code reviews, allowing multiple developers to provide feedback, suggest improvements, and discuss code changes collaboratively.
- **Technical Interviews:** Candidates can use real-time coding platforms to participate in technical interviews, where they are evaluated based on their coding skills and problem-solving abilities in a collaborative environment.
- **Teaching and Learning** Educators can leverage real-time coding platforms to facilitate interactive coding lessons and workshops, allowing students to collaborate on coding exercises and projects in real time.

## 2.6 Software Resources Required

1. **Web Framework:** A web framework is necessary for developing the backend logic.
2. **Programming Languages:** Programming languages such as JavaScript, HTML and CSS are essential for developing different components of the Live Coder application, including backend logic and frontend interfaces.
3. **Version Control System:** Version control software such as Git is necessary for managing the application's source code, tracking changes, and collaborating with team members.

# Chapter 3

# Software requirement specification

## 3.1 Introduction

### 3.1.1 Purpose

The purpose of "a Live Coder" is to create an inclusive and versatile collaborative coding platform, enabling real-time collaboration among users working with languages like C. This system seeks to facilitate seamless code sharing and editing, incorporating interactive coding features for immediate testing. With integrated video calling, it enhances communication during collaborative sessions, ensuring efficient discussions and problem-solving. The inclusion of version control and a user-friendly interface aims to provide stability, security, and an intuitive experience for users of varying skill levels. Platform agnosticism allows accessibility from different devices, while scalability ensures adaptability to a growing user base and evolving project demands, making "a Live Coder" a comprehensive solution for collaborative programming endeavors.

### 3.1.2 Intended Audience and Reading Suggestions

The document is designed to cater to a diverse group of individuals involved in the development of a software tool or platform called "Live Coder." The intended audience encompasses interviewers, developers, project managers, and any other stakeholders who play a role in the development process.Also this would be very helpful for programming enthusiasts. To ensure that the document serves the needs of this varied audience, it is recommended that readers commence their exploration by delving into the overview sections. These introductory sections are crafted to provide a broad understanding of the Live Coder project, offering a high-level perspective on its goals, functionalities, and overarching objectives.

### 3.1.3 Use-cases

- **Collaborative Coding** Users create collaborative session to code efficiently.
- **Messaging:** Users can message while coding in case of lag in editor or audio.
- **Video Calling:** Users can engage in video and audio call.
- **Syntax:** Syntax is highlighted for C-like languages.

### 3.1.4 Product Scope

The "a-Live Coder" is a sophisticated web-based application designed to facilitate real-time collaborative coding experiences. Developed using a stack of technologies that includes Javascript, HTML, CSS, WebRTC, and CodeMirror, the platform leverages a combination of cutting-edge tools to deliver a seamless and dynamic coding environment

## 3.2 Specific Requirements:

### 3.2.1 Performance Requirements

- **Response Time:** The system should respond to user actions (login, product selection, price comparison) within a reasonable time frame.
- **Scalability:** The system should be able to handle an increasing number of users and products without a significant decrease in performance. This involves efficient use of server resources and possibly implementing load balancing techniques.
- **Concurrency:** The system should support multiple users accessing and interacting with the application simultaneously without experiencing performance degradation. This includes handling concurrent requests for web scraping data from multiple websites.

### 3.2.2 Safety and Security Requirements:

- **Data integrity:** HTTPS which is an extension of HTTP, provides security through the use of encryption protocols such as SSL/TLS (Secure Sockets Layer/Transport Layer Security). HTTPS encrypts data transmitted between the clients, ensuring confidentiality, integrity, and authenticity of the communication.

## 3.3 System Architecture

### 3.3.1 Client-Side Application:

- Client-side application is the user-facing interface where users interact with the platform. It includes the code editor, video calling interface and messaging.

  **Technologies:** HTML, CSS, JavaScript (for web applications), or platform-

specific frameworks for desktop applications.

### 3.3.2 Real-Time Collaboration Mechanism:

* component manages real-time collaboration features, including syncing code changes across multiple clients, handling concurrent edits, and resolving conflicts.

  **Technologies:** WebSocket protocol, WebRTC for video calling, and libraries/frameworks like Socket.io for real-time communication and code Mirror for editor.

## 3.4 Design Constraints

– **Data acquisition**

  * **Website structure and its changes:** Adapting to variations in website layouts and data formats.
  * **Security:** The techniques must comply with website terms of service and keep in mind data privacy regulations.

– **Client-side constraints**

  * **Synchronization of Data:** We need to ensure the consistency of video and changes in the editor on client-side in case of many users.

# Chapter 4

# Detailed Design Document

## 4.1 Data Flow Diagram

### 4.1.1 DFD-level 0

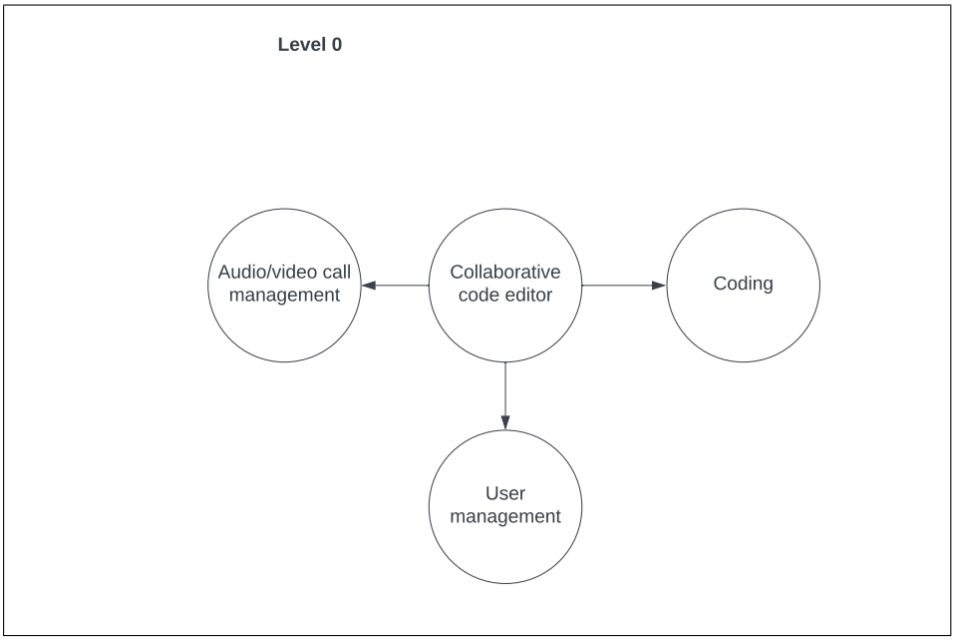


Figure 4.1: dfd-0 diagram

### 4.1.2 DFD-level 1

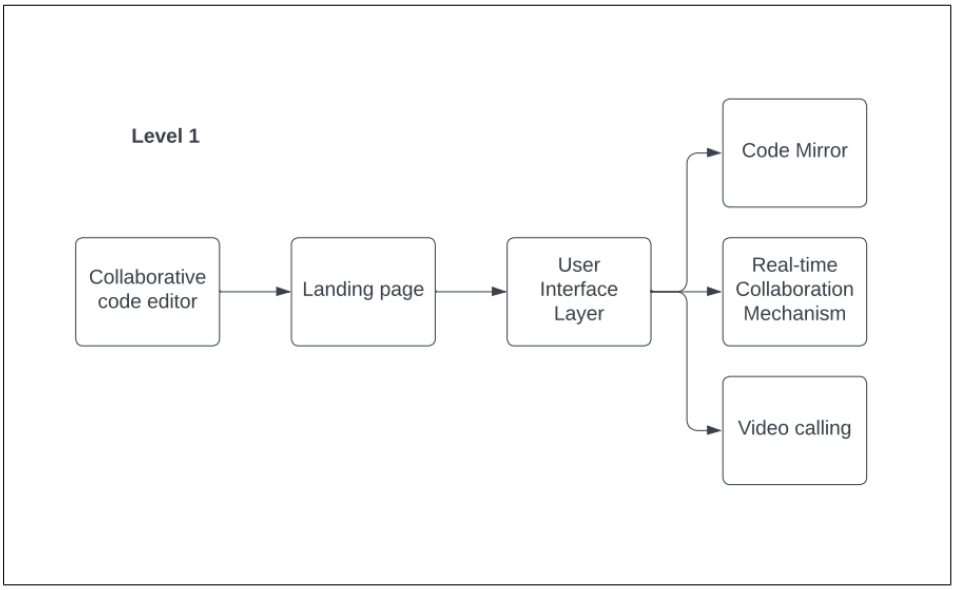


Figure 4.2: dfd-1 diagram

### 4.1.3 DFD-level 2

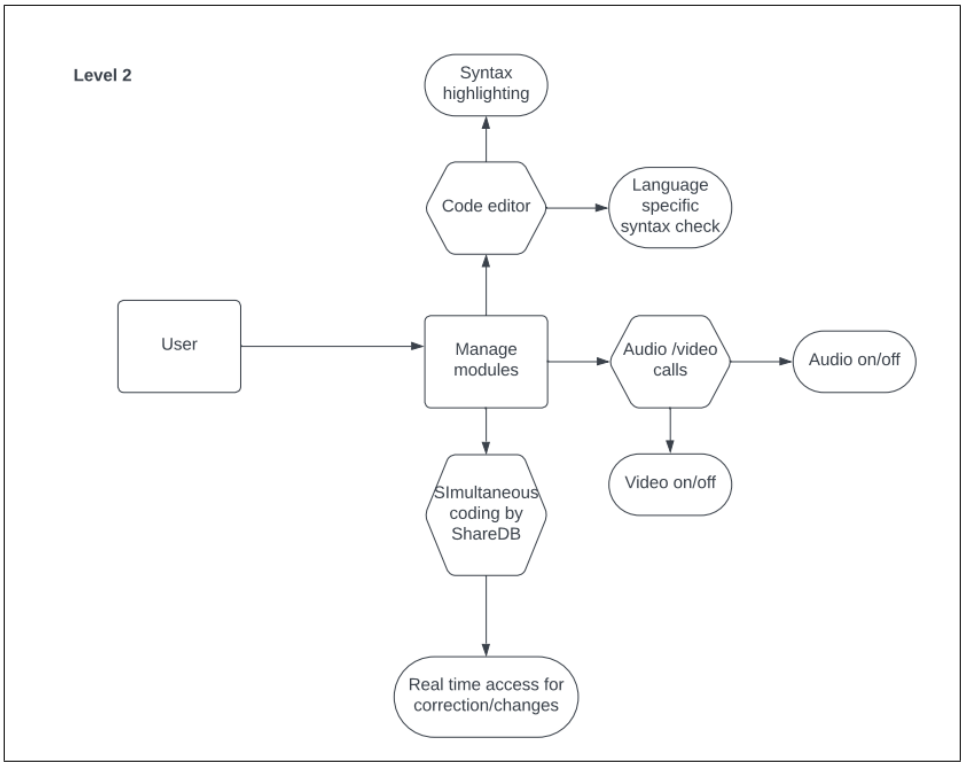


Figure 4.3: dfd-2 Diagram

## 4.2 Component Design

### 4.2.1 Class Diagram

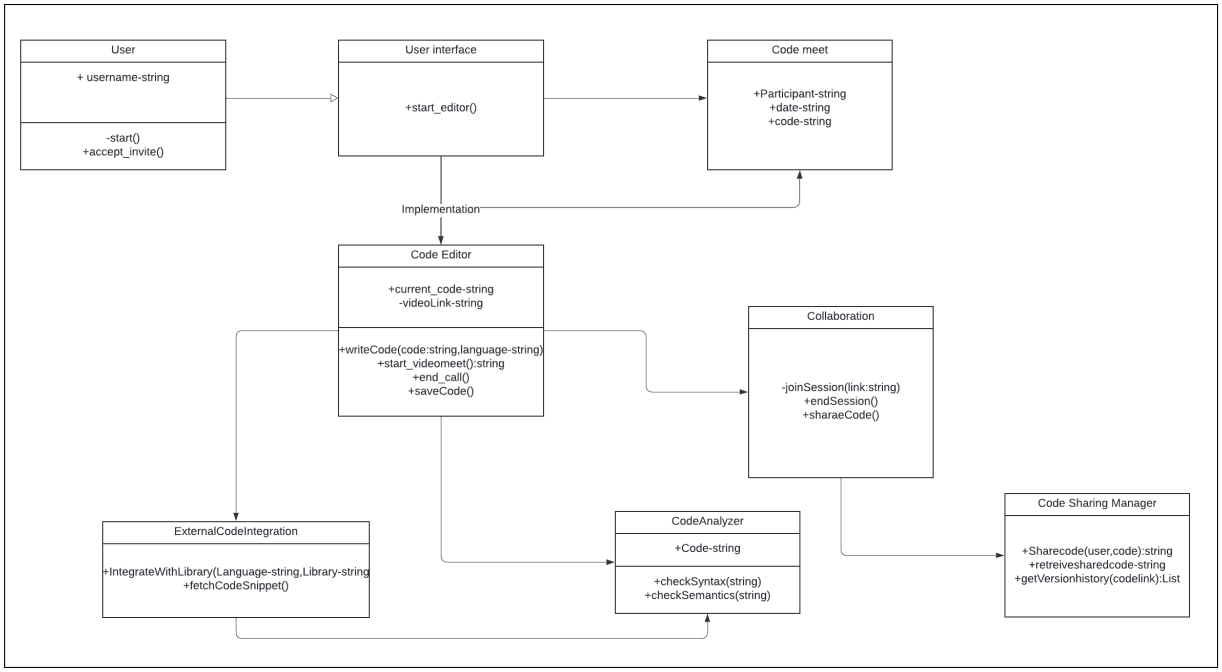


Figure 4.4: Class Diagram

### 4.2.2 Activity Diagram

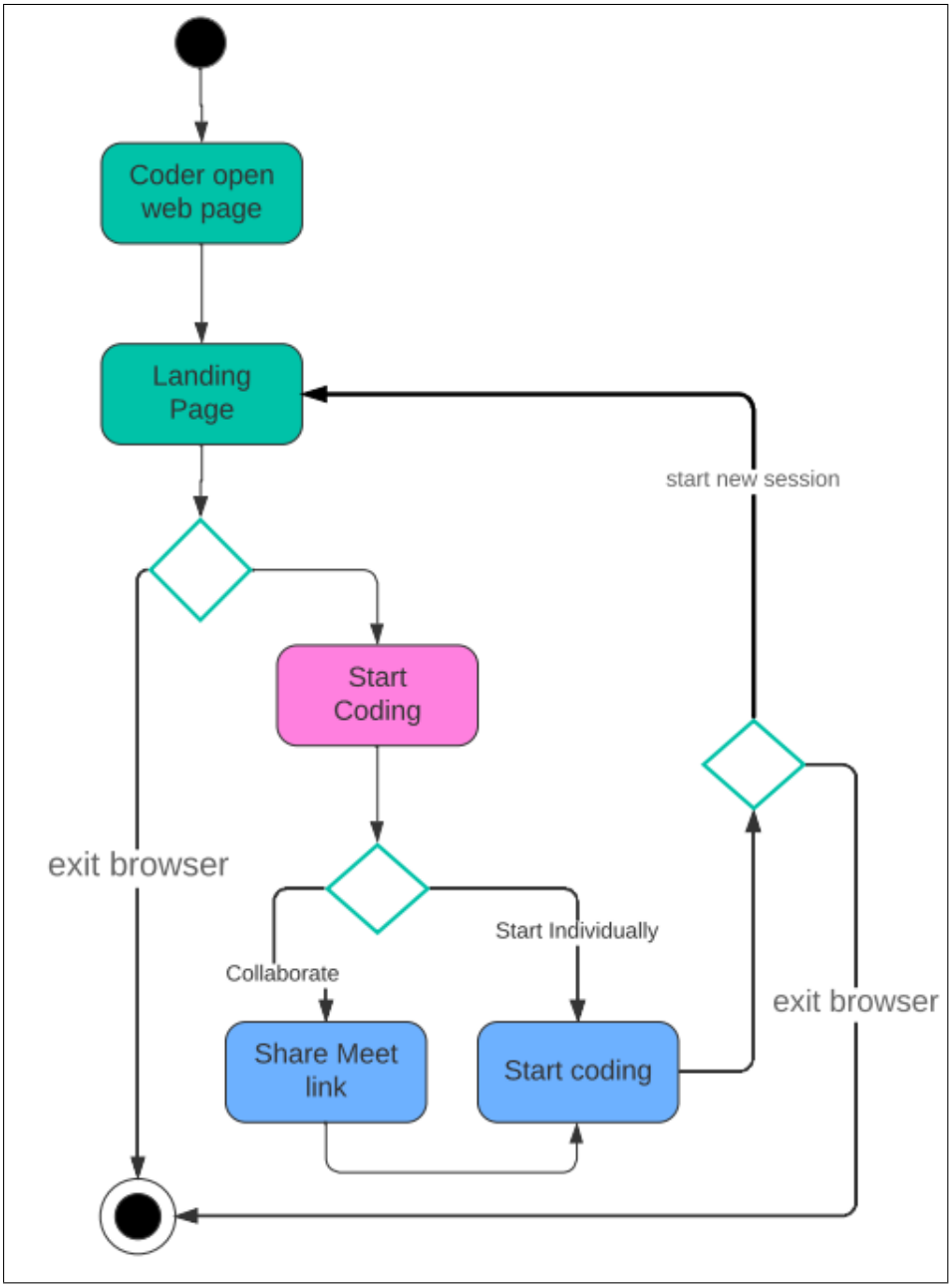


Figure 4.5: Activity Diagram

# Chapter 5

# Implementation and testing

## 5.1 Code snippet

```
io.on("connection", socket => {
    console.log('socket connected..', socket.id);

    socket.on('content_change', (data) => {
        const room = data.documentId;
        socket.to(room).emit('content_change', data.changes);
    });

    socket.on('register', function(data) {
        const room = data.documentId;
        socket.nickname = data.handle;
        socket.join(room);
        let members = [];
        for (const clientId of io.sockets.adapter.rooms.get(room)) {
            members.push({
                id: clientId,
                name: io.sockets.sockets.get(clientId).nickname
            });
        }
        console.log(members);
        io.in(room).emit('members', members);
        socket.to(room).emit('register', { id: socket.id, name: data.handle });
    });

    socket.on("join-room", (roomId, userId, userName) => {
        console.log(roomId, userId, userName);
        socket.join(roomId);
        socket.broadcast.to(roomId).emit("user-connected", userId);
    });
```

Figure 5.1: socket code snapshot

```
    socket.on("message", (data) => {
        io.to(data.id).emit("createMessage", data.message, data.name);
        // console.log(message);
    });

    socket.on('disconnect', function(data) {
        socket.broadcast.emit('user_left', { id: socket.id });
    });
});
```

Figure 5.2: socket code snapshot 2

```
function fastDiff(a, b, cmp, atomicChanges = false) {
    cmp = cmp || function(a, b) {
        return a === b;
    };

    if (!Array.isArray(a)) {
        a = Array.prototype.slice.call(a);
    }

    if (!Array.isArray(b)) {
        b = Array.prototype.slice.call(b);
    }
    const changeIndexes = findChangeBoundaryIndexes(a, b, cmp);
    return atomicChanges ? changeIndexesToAtomicChanges(changeIndexes, b.length) : changeIndexesToChanges(b, changeIndexes);
}

function findChangeBoundaryIndexes(arr1, arr2, cmp) {
    const firstIndex = findFirstDifferenceIndex(arr1, arr2, cmp);

    if (firstIndex === -1) {
        return { firstIndex: -1, lastIndexOld: -1, lastIndexNew: -1 };
    }

    const oldArrayReversed = cutAndReverse(arr1, firstIndex);
    const newArrayReversed = cutAndReverse(arr2, firstIndex);

    const lastIndex = findFirstDifferenceIndex(oldArrayReversed, newArrayReversed, cmp);

    const lastIndexOld = arr1.length - lastIndex;
    const lastIndexNew = arr2.length - lastIndex;

    return { firstIndex, lastIndexOld, lastIndexNew };
}
```

Figure 5.3: code for reflecting changes

## 5.2 UI implementation

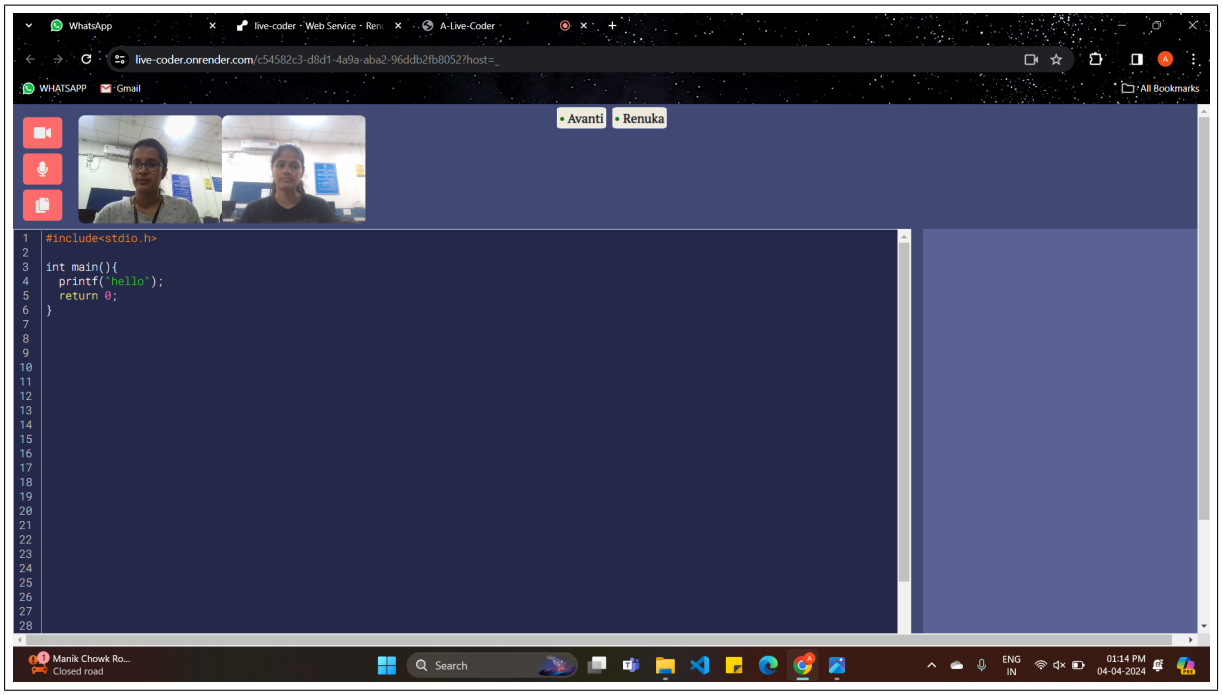


Figure 5.4: Editor working

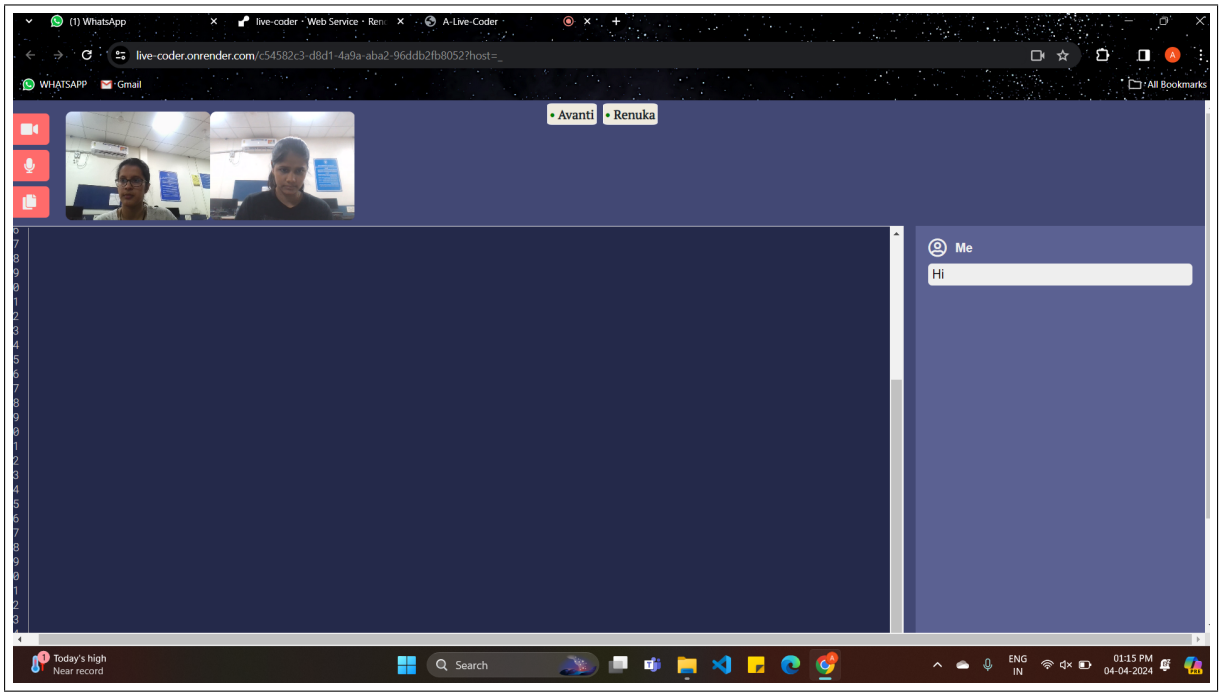


Figure 5.5: Message working

## 5.3 Testing

### 5.3.1 White Box Testing

#### Test Case 1: Socket connection

**Test:**

1. Test event emission and handling: Ensure that events are emitted and received correctly between clients and the server.
2. Test broadcasting: Verify that messages or events are broadcasted to the appropriate recipients.

**Expected Result:**

– Proper emission and broadcasting.

#### Test Case 2: Room Management

**Test:**

1. Test room creation: Ensure that new rooms can be created successfully.
2. Test joining and leaving rooms: Validate that users can join and leave rooms as expected.

**Expected Result:**

– The system could properly create rooms with unique room IDs and when the user joined or left the room, the name disappears, meaning the user has been removed from the room correctly.

#### Test Case 3: Concurrency and Scalability

1. Test under load: Simulate high traffic conditions to assess how the application performs under load.
2. Test scalability: Evaluate how the application scales with increasing numbers of users or connections.

**Expected Result:**

– The system can handle multiple users but gives a lag when more than 3 users enter the room. We can conclude that there are scalability issues which we will work on in further versions.

### 5.3.2 Black Box Testing

#### Test Case 1: User Interface Testing

**Test:**

1. Validate the usability and responsiveness of the user interface.
2. Test navigation through the application to ensure a seamless user experience.

**Expected Result:**

– The system worked seamlessly.

#### Test Case 2: Functional Testing

**Test:**

1. Test the core functionalities of the application, such as collaborative code editing, video calling, and messaging.

2. Verify that users can perform actions like creating rooms, joining calls, sending messages, and editing code without encountering errors.

**Expected Result:**

– The system should accurately work these features with proper working of audio/video.

# Chapter 6

# Conclusion and Future Scope

In conclusion, the development of a collaborative editor with integrated video calls, audio calls, and messaging functionalities represents a significant advancement in remote teamwork and productivity tools. By allowing multiple users to code together in real-time while communicating via various channels, such a platform streamlines collaboration and facilitates rapid idea exchange. This enhances project management efficiency and fosters a sense of camaraderie among remote team members. The seamless integration of coding and communication features not only accelerates the development process but also promotes engagement and cohesion within distributed teams. Overall, the collaborative editor serves as a valuable asset for modern organizations seeking to optimize their remote workflows and maximize productivity.

Looking ahead, there are several avenues for further enhancing the collaborative editor to meet evolving user needs and technological advancements. Firstly, incorporating advanced collaboration features like real-time code execution and code debugging tools can streamline the development process and improve code quality. Additionally, integrating with popular project management tools will enable seam-

less transition between planning and execution stages. AI-driven assistance, such as code suggestions and error detection, holds promise for enhancing productivity and code quality. Moreover, customization options for the editor interface and themes can cater to diverse user preferences, while mobile accessibility will extend the platform's reach to users on the go. Security enhancements and scalability improvements will be crucial for gaining trust and accommodating a growing user base. Finally, integration with educational platforms can foster collaborative learning environments and provide valuable resources for aspiring developers. By addressing these future scope areas, the collaborative editor can continue to evolve as an indispensable tool for remote collaboration and software development, empowering teams to work efficiently regardless of geographical constraints.