Hindustan Institute of Technology and Science

Code Crunch 2024

(National Level Hackathon)

Low-Level Design

on

AI-Driven Real-time Collaboration tool - Code Editor

Team members:
Yogeshwar C M (TL)
Acsah Susan Mathew
Tharun Raju
Sachin Kumar
Kanchana Krishna

Table of Contents

1. Introduction

1.1. Scope of the Document

This document outlines the design and development of a collaborative, AI-enhanced real-time online code editor. The editor is built to support multiple programming languages: Python, Java, C++, C, and JavaScript. It provides a collaborative coding environment with real-time code editing and features advanced AI capabilities.

The scope of this document includes:

- An overview of the system architecture and components
- Detailed design and implementation strategies
- Mechanisms for data management and validation
- The frameworks and libraries used in development
- Strategies for unit testing and quality assurance

1.2. Intended Audience

This document is intended for:

- **Developers**: To provide guidance on the system's design and integration, aiding in development and future enhancements.
- **Project Managers**: To offer insights into the system's structure and progress, ensuring project goals are met.
- **Technical Reviewers**: To assess the design and implementation for quality and compliance with technical standards.
- **Quality Assurance Engineers**: To understand the testing framework and validate the system's functionality.

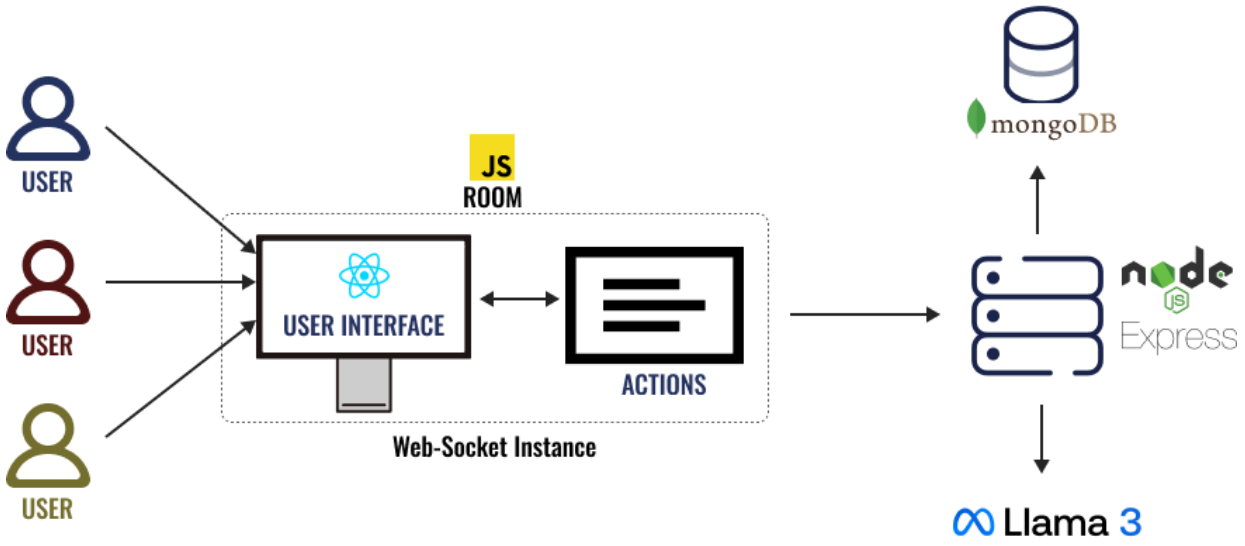
1.3. System Overview

The collaborative code editor allows users to write and edit code in real-time, with support for several programming languages. It leverages modern technologies including React for the front-end, Node.js and Express for the back-end, and MongoDB for data storage. Real-time interactions are facilitated through WebSocket technology, and AI features are integrated using Ollama to provide coding suggestions and improvements.

2. Low Level System Design

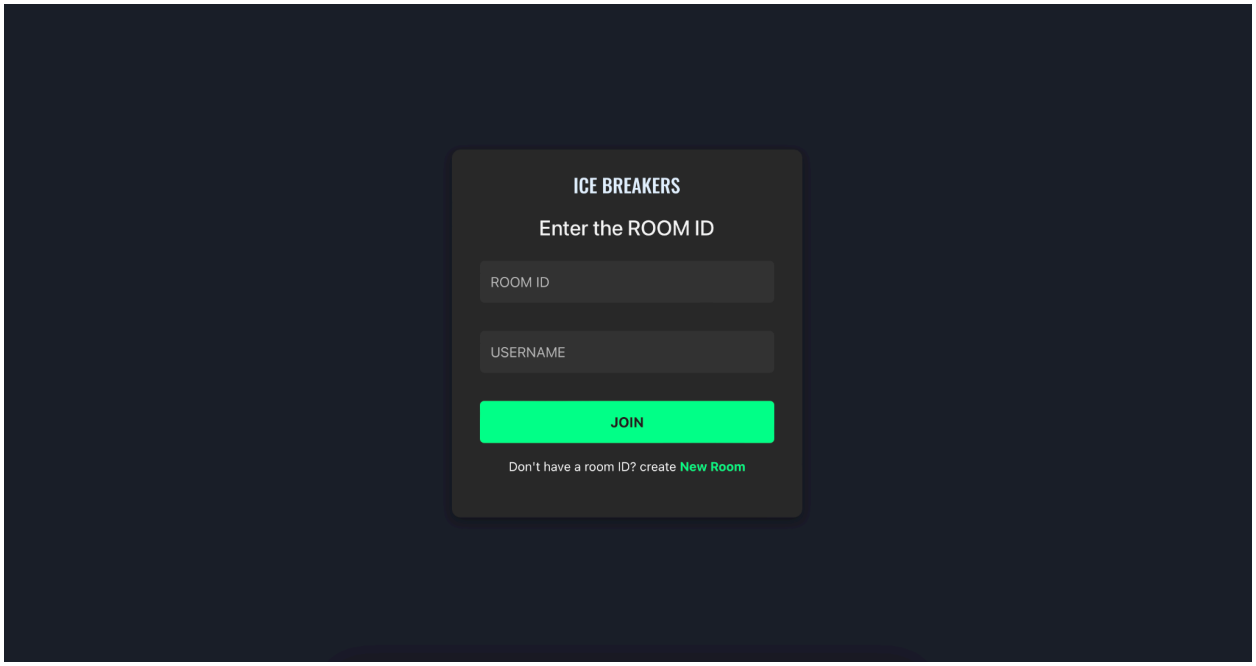
2.1. Sequence Diagram

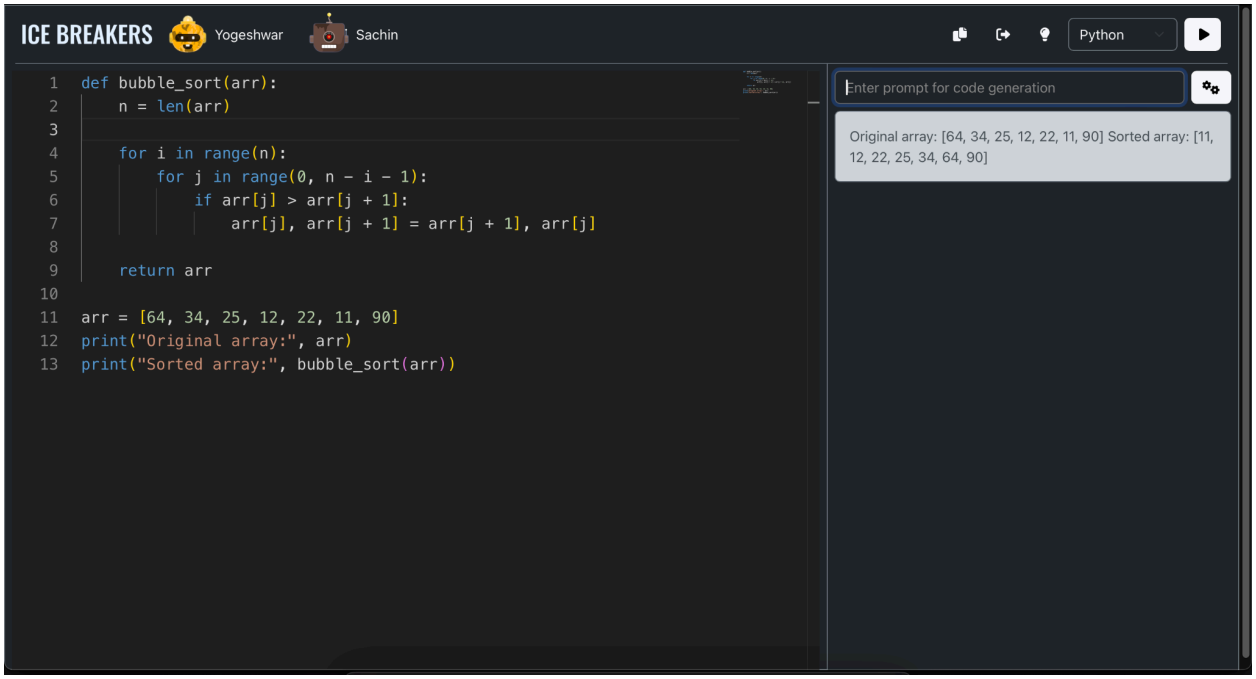
Below is the diagram illustrating the sequence of interactions between various components during key operations.



2.2. Navigation Flow/UI Implementation

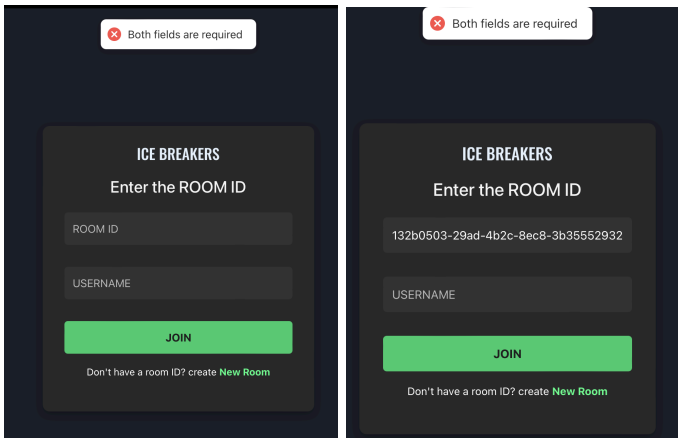
- **Home Page**: Users can either create a new room or join an existing room using any username (nickname).
- **Editor Page**:
  - **Navigation Bar**: Located at the top, includes:
    - List of members in the room
    - Options to leave the room
    - Copy the room code
    - Get AI suggestions for code
    - Compile code
    - Select language from a dropdown menu
  - **Editor Area**: Located below the navigation bar, provides a real-time collaborative environment for coding using the Monaco Editor.
  - **Right Pane**: Includes:
    - An AI prompt field for generating code with AI assistance
    - A compilation result window displaying the output


ICE BREAKERS
Enter the ROOM ID
ROOM ID
USERNAME
JOIN
Don't have a room ID? create New Room


ICE BREAKERS
Yogeshwar
Sachin
Python
def bubble_sort(arr):
n = len(arr)
for i in range(n):
for j in range(0, n - i - 1):
if arr[j] > arr[j + 1]:
arr[j], arr[j + 1] = arr[j + 1], arr[j]
return arr
arr = [64, 34, 25, 12, 22, 11, 90]
print("Original array:", arr)
print("Sorted array:", bubble_sort(arr))
Enter prompt for code generation
Original array: [64, 34, 25, 12, 22, 11, 90] Sorted array: [11, 12, 22, 25, 34, 64, 90]

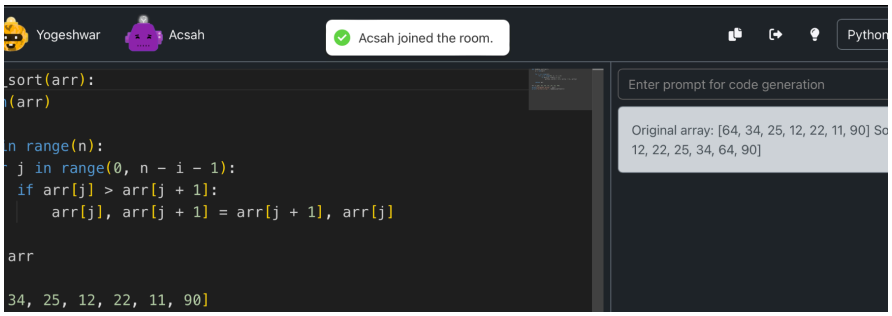
2.3. Screen Validations, Defaults, and Attributes

- **Home Screen**: Validates that the room code and username fields are filled before allowing users to join a room. If either field is empty, a toast notification is displayed to the user.



2.4. Client-Side Validation Implementation

- Displays user-friendly toast notifications and confirmation messages for various actions, such as:
  - A user joining or leaving the room
  - Copying the room code
  - Handling socket connection issues due to network problems



2.5. Server-Side Validation Implementation

- Implements error handling using try-catch blocks for asynchronous operations and API responses.
- Processes output received from Ollama to extract and forward only the relevant code to the client side.

2.6. Components Design Implementation

- **Monaco Editor**: Configured to support five programming languages with the VS-Dark theme.
- **Ollama Model**: Operates in a separate runtime for AI integration, providing code generation and assistance features.

2.7. Configurations/Settings

- **Font**: The Lato font is used for text outside the Monaco Editor.
- **WebSocket Actions**:
  - JOIN: “join”
  - JOINED: “joined”
  - DISCONNECTED: “disconnected”
  - SYNC_CODE: “sync-code”
  - CODE_CHANGE: “code-change”
  - COMPILE_CODE: “compile-code”
  - RECOMMEND_CODE: “recommend_code”
  - RECOMMEND_RESULT: “recommend-result”
  - COMPILATION_RESULT: “compilation-result”
  - GENERATE_CODE: “generate-code”
  - CODE_GENERATION_RESULT: “code-generation-result”
- **Ollama Version**: Ollama 3.1 is used locally for AI-related assistance and code generation.

2.8. Interfaces to Other Components

- **Client-Side**: Establishes a new WebSocket connection from the backend server based on the room ID.
- **Backend**: Maintains a MongoDB record of the latest code when users leave the room.
- **Real-Time Collaboration**: Utilizes WebSocket actions like SYNC_CODE and CODE_CHANGE to keep the code synchronized across all clients.
- **AI Assistance**: The Ollama Node module is used for providing AI-related prompts and code suggestions.

3. Data Design

3.1 List of Key Schemas/Tables in Database

Code Schema

```
const mongoose = require("mongoose");

const codeSchema = new mongoose.Schema({
  roomId: { type: String, required: true, unique: true },
  code: { type: String, default: "" },
  createdAt: { type: Date, default: Date.now, expires: "7d" },
});

const Code = mongoose.model("Code", codeSchema);

module.exports = Code;

```

- **roomId**: This is a string field that uniquely identifies a room. It is marked as required and unique to ensure that each room has a distinct identifier in the database.
- **code**: This field holds the code that users are collaboratively editing. It is a string and has a default value of an empty string. This allows the field to be optional, but will store the code when it is available.
- **createdAt**: A date field that records when the document was created. It defaults to the current date and time (Date.now) and is set to expire 7 days after creation. This automatic expiration helps in cleaning up old records that are no longer needed.

This schema is represented by the Mongoose model Code which interacts with MongoDB to handle CRUD operations for the code data associated with different rooms.

3.2 Details of Access Levels on Key Tables in Scope

- **Code Collection**: The Code schema's access levels are controlled through your application's business logic and authorization mechanisms. Generally, access is restricted as follows:
  - **Create**: Authorized users (e.g., those who have joined a room) can create or update code entries.
  - **Read**: Users within the room can access and view the code.
  - **Update**: Users with edit permissions can modify the code.
  - **Delete**: Code entries will automatically be deleted after 7 days from creation. Manual deletion can be performed by authorized admin users if needed.

3.3 Key Design Considerations in Data Design

- **Data Consistency**: Ensuring that each room's code is synchronized across all clients in real-time. This involves updating the code field in the database whenever changes occur.
- **Expiration Policy**: Implementing an expiration policy with the createdAt field to automatically clean up old records. This helps in managing database size and ensuring only relevant data is kept.
- **Indexing**: The roomId field is indexed as unique to quickly retrieve code for a specific room and to ensure there are no duplicate room entries.
- **Scalability**: As the number of rooms and users grows, the schema should efficiently handle increased load and data volume. Using Mongoose and MongoDB's built-in scalability features will help manage this growth.
- **Security**: Ensuring that code data is not exposed to unauthorized users by implementing proper authentication and authorization checks.

4. Details of Other Frameworks Being Used

In this section, we outline the key frameworks, libraries, and tools utilized in the project, highlighting their roles and functionalities within the system.

4.1 Frontend Libraries and Frameworks

- **React.js**:
  - The frontend is built using React, a popular JavaScript library for building user interfaces. It allows for a component-based architecture, making the code more modular, maintainable, and scalable.
  - **React Bootstrap**: Used for styling components and implementing responsive design. It provides a collection of reusable UI components that follow the Bootstrap framework, helping to create a consistent and visually appealing user interface.
- **Monaco Editor**:
  - The collaborative code editor is built using the Monaco Editor, which is the same editor that powers Visual Studio Code. It provides syntax highlighting, IntelliSense, and code formatting for multiple programming languages (Python, Java, C++, C, JavaScript).
  - The editor is integrated into the React application to support real-time collaborative editing, leveraging WebSockets for seamless interaction.

4.2 Backend Libraries and Frameworks

Node.js and Express.js:

- The backend server is implemented using Node.js with Express.js as the web framework. Express simplifies routing, middleware integration, and handling of HTTP requests, forming a robust backend structure for managing WebSocket connections and server-side logic.
- **Cors**: Handles Cross-Origin Resource Sharing issues, ensuring secure and seamless communication between the client and server, especially during development.

Socket.IO:

- **Socket.IO** is utilized for real-time, bi-directional communication between the client and server. It allows for efficient event-based communication, essential for the collaborative editing features of the application.
- Key features handled by Socket.IO include user actions such as joining/leaving rooms, synchronizing code changes, sending AI suggestions, and compiling code. This library ensures robust real-time collaboration capabilities across the application.

4.3 AI Integration Framework

Ollama for AI Integration:

- Ollama is used for integrating AI-driven code suggestions and generation features. It provides a local runtime environment to run AI models for natural language processing tasks.
- The backend interacts with Ollama's models to provide intelligent suggestions and code completions based on user input, enhancing the user experience with context-aware AI-powered features.

4.4 Database and Data Management

- **MongoDB**:
  - MongoDB is used as the primary database to store room-specific code and user interactions. It allows storing documents in a flexible, schema-less format, making it ideal for managing dynamic and evolving data models.
  - The data model for storing room-specific code is designed to expire after a set duration (e.g., 7 days) to prevent storage bloat and ensure efficient data management.

5. Unit Testing

Unit testing for WebSocket actions was conducted using Jest, Supertest, and socket.io-client. Here's a brief overview of the results:

- **JOIN**: Verified that clients successfully join rooms and receive updates.
- **JOINED**: Ensured new clients are notified to others in the room.
- **DISCONNECTED**: Confirmed the server handles disconnections and notifies remaining clients.
- **SYNC_CODE**: Validated that code changes synchronize correctly between clients.
- **CODE_CHANGE**: Checked if code updates are broadcasted to all clients.
- **RECOMMEND_CODE**: Verified code improvement recommendations are processed and returned.
- **COMPILE_CODE**: Ensured code compilation results are correctly processed and sent back.
- **RECOMMEND_RESULT**: Confirmed recommendations are provided on request.
- **COMPILATION_RESULT**: Validated that compilation results and errors are returned properly.
- **GENERATE_CODE**: Tested successful generation of code based on a prompt.
- **CODE_GENERATION_RESULT**: Ensured generated code results are returned accurately.

6. Key Notes

- **Real-Time Collaboration:** Utilizes WebSockets (Socket.IO) for real-time code editing and synchronization.
- **AI Integration: Incorporates** Ollama AI for code suggestions and generation.
- **Technology Stack:** Built with MERN stack, Tailwind CSS, React Bootstrap, and Monaco Editor.
- **Session Management:** Not explicitly implemented; relies on WebSocket connections and MongoDB for state management.
- **Caching:** Not utilized; focuses on real-time interactions.
- **Unit Testing:** Conducted with Jest and React Testing Library to ensure functionality and reliability.
- **Code Management:** MongoDB schema tracks code per room, with automatic expiration after 7 days.
- **User Experience:** Features user-friendly toasts and confirmation messages for a smooth interaction.
- **Future Improvements:** Plans include enhancing AI capabilities and refining the UI.

7. References

R. W. Hamming, “Error detecting and error correcting codes,” *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147-160, April 1950.

V. Pimentel and B. G. Nickerson, “Communicating and Displaying Real-Time Data with WebSocket,” *IEEE Internet Computing*, vol. 16, no. 4, pp. 45-53, July-Aug. 2012.

P. Srivani, S. Ramachandram, and R. Sridevi, “A survey on client side and server side approaches to secure web applications,” in *2017 International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, Coimbatore, India, 2017, pp. 22-27.

Dr. Santosh Kumar Shukla, Shivam Dubey, Tarun Rastogi, and Nikita Srivastava, “Application using MERN Stack,” *International Journal for Modern Trends in Science and Technology*, vol. 8, no. 06, pp. 102-105, 2022.

C. Győrödi, R. Győrödi, G. Pecherle, and A. Olah, “A comparative study: MongoDB vs. MySQL,” in *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, Oradea, Romania, 2015, pp. 1-6.

S. Tilkov and S. Vinoski, “Node.js: Using JavaScript to Build High-performance Network Programs,” *IEEE Internet Computing*.

L. Sinanaj, J. Ajdari, M. Hamiti, and X. Zenuni, “A comparison between online compilers: A Case Study,” in *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro, 2022, pp. 1-6.

J. Ferreira, J. Noble, and R. Biddle, “Agile Development Iterations and UI Design,” in *Agile 2007 (AGILE 2007)*, Washington, DC, USA, 2007, pp. 50-58, doi: 10.1109/AGILE.2007.8.

J. Lu, L. Yu, X. Li, L. Yang, and C. Zuo, “LLaMA-Reviewer: Advancing Code Review Automation with Large Language Models through Parameter-Efficient Fine-Tuning,” in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, Florence, Italy, 2023, pp. 647-658.

B. S. Rawal, L. Berman, and H. Ramcharan, “Multi-client/Multi-server split architecture,” in *The International Conference on Information Networking 2013 (ICOIN)*, Bangkok, Thailand, 2013, pp. 696-701.