Software Requirements Specification (SRS) for Real-time Collaborative Online IDE

(IEEE 830-Style Format)

1. Introduction

This document provides a comprehensive Software Requirements Specification (SRS) for the **Real-time Collaborative Online Integrated Development Environment (RCO-IDE)**. It serves as the architectural blueprint for the development and deployment of a MERN-based platform designed to facilitate synchronous collaborative programming.

1.1 Purpose

The manual coordination of collaborative programming tasks is inefficient, often relying on screen-sharing or asynchronous code exchange. This introduces environment configuration discrepancies and security vulnerabilities. The **Automated Collaborative System** eliminates these frictions by providing an instant, cloud-native coding workspace. This system leverages WebSocket protocols and containerized sandboxing to ensure a secure, real-time "multiplayer" coding experience, specifically tailored for **JavaScript** and **Python** development.

1.2 Document Conventions

- **FR-X:** Functional Requirement.
- **NFR-X:** Non-Functional Requirement.
- **Must/Shall:** Mandatory requirements.
- **Should/May:** Desirable requirements.

1.3 Intended Audience

- **Project Stakeholders:** To understand the system objectives and business value.
- **System Designers:** To analyze requirements for the MERN stack implementation.
- **DevOps/Security Engineers:** To understand containerization (Docker) and isolation strategies.
- **QA Engineers:** To derive test cases for validation.

1.4 Project Scope

The System provides an online interface where users can register and access a synchronized coding environment.

- **Session Host (The Authority):** Automates environment setup for interviews or teaching, reducing workload.

- **Communication Platform:** Facilitates real-time transmission of keystrokes and operational transforms between Guest and Host.
- **Code Execution:** securely transfers code to a backend execution engine (Judge0) for compilation and verification.
- **Language Support:** Strictly supports **JavaScript** and **Python** runtimes.

Exclusions: Persistent hosting of long-running applications and complex build chains for compiled languages.

1.5 References

- IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.
- Judge0 API Documentation.
- Socket.IO Documentation.
- Microsoft Monaco Editor Documentation.
- MongoDB Manual: Data Modeling & Schema Design.

2. Overall Description

2.1 Product Perspective

The RCO-IDE interfaces between the 'Applicant' (Guest) and the 'Administrator' (Host). It replaces local IDEs with an ephemeral, browser-based solution. The system functions as a client for the browser (Presentation Layer) and a client for the Execution Engine (Integration Layer). It operates independently of the user's local machine configuration, ensuring consistent behavior across different devices.

2.2 Product Functions

- **Secure Registration:** Users create accounts to maintain session history, authenticated via JWT.
- **Real-time Communication:** A dedicated chat and code synchronization channel using WebSockets.
- **Code Execution Reports:** The Administrator can trigger code runs, with the system managing the lifecycle of submission, compilation, and output streaming.
- **Synchronized Editing:** Concurrent modification of the text buffer with conflict resolution (CRDTs/OT).
- **Session State Management:** Persistence of code state to allow reconnection without data loss.

2.3 User Classes and Characteristics

- **Applicant (Guest):** Accesses the system via a shared link. Has read/write access to the editor but limited control over session settings.
- **Administrator (Host):** Initiates sessions, controls language settings (JS/Python), and manages permissions (e.g., Read-Only mode). Responsible for the "issuance" of the coding challenge.

- **Police (Verification System / Judge0):** An automated system actor that validates code logic, checks for syntax errors, and enforces security constraints (e.g., preventing infinite loops).
- **Database Administrator:** Manages user data and system logs within the MongoDB cluster.

2.4 Operating Environment

- **Web Interface:** Accessible via modern browsers (Chrome, Firefox, Edge, Safari).
- **Backend Infrastructure:** Cloud-hosted Node.js runtime (v18+).
- **Database:** MongoDB for persistent storage of user and session data.
- **Connectivity:** Requires HTTPS/WSS (WebSockets) for real-time functionality.
- **Containerization:** The execution engine runs in a Linux environment with Docker for strict sandboxing.

2.5 Design and Implementation Constraints

- **Hardware:** Client devices require sufficient RAM (4GB+) for rendering the Monaco Editor. Server requires CPU resources for Docker container management.
- **Security:** Remote Code Execution (RCE) mandates strict network isolation and read-only file systems within the sandbox to prevent server compromise.
- **Latency:** Real-time collaboration requires low latency (<300ms RTT) for a seamless typing experience.
- **Language Support:** constrained to JavaScript and Python to simplify compiler configuration.

2.6 User Documentation

- **Interactive Tour:** Onboarding overlay highlighting key features.
- **API Documentation:** Internal specs for REST endpoints.
- **Admin Guide:** Configuration for self-hosted Judge0 instances.

2.7 Assumptions and Dependencies

- **Literacy:** Users possess basic computer and English language skills.
- **Service Availability:** System operation depends on the uptime of the MongoDB cluster and Judge0 API.
- **Browser Support:** Users utilize browsers supporting WebSockets and ES6 standards.

3. System Features (Functional Requirements)

3.1 User Registration & Login

- **FR1:** The system shall allow users to register with unique credentials (email/phone), validating formats and enforcing password complexity.
- **FR2:** The system shall manage secure login/logout using stateless **JSON Web Tokens (JWT)**.

- **FR3:** The system shall provide a mechanism for password recovery via email tokens.

3.2 Application Management (Session Creation)

- **FR4:** The system shall allow users to create new coding sessions or rejoin existing ones, initializing a document in MongoDB.
- **FR5:** The system shall allow users to configure session parameters, including Name, Default Language (Python/JS), and Privacy Mode.
- **FR6:** The system shall enable file uploads (e.g., input text files) to be accessible by the code during execution.

3.3 Appointment & Scheduling (Collaboration Invitation)

- **FR7:** The system shall generate unique, obfuscated URLs (UUID-based) for inviting peers to a session.
- **FR8:** The system shall allow the Host to revoke links or close sessions to prevent unauthorized future access.

3.4 Payment Processing (Resource Allocation)

- **FR9:** The system shall track resource usage (CPU time) for every code execution, enforcing rate limits to prevent abuse.
- **FR10:** The system shall provide a real-time console log acting as a "receipt" of execution, displaying standard output or error messages.

3.5 Verification & Validation (Code Compilation)

- **FR11:** The system shall forward source code and standard input to the verification authority (Judge0) via secure API requests.
- **FR12:** The system shall interpret verification statuses returned by the sandbox (e.g., Accepted, Syntax Error, Time Limit Exceeded).
- **FR13:** The system shall immediately notify the client of the execution result via WebSocket events.

3.6 Status Tracking (Real-time Synchronization)

- **FR14:** The system shall synchronize document state across all connected clients using Operational Transformation or CRDTs, ensuring eventual consistency.
- **FR15:** The system shall provide transient notifications for user join/leave events and execution status changes.

3.7 Passport Issuance (Code Export & Snapshot)

- **FR16:** The system shall allow users to download the current codebase as a local file (e.g., .py or .js).
- **FR17:** The system shall allow Administrators to save timestamped snapshots of the code to the database for future retrieval.
- **FR18:** The system shall maintain a history log, allowing users to revert to previous code

states.

3.8 Administrative Functions

- **FR19:** The system shall enforce Role-Based Access Control (RBAC):
    - **Host:** Full control (Edit, Run, Kick, Settings).
    - **Guest:** Edit and Run privileges only.
    - **Observer:** Read-only access.
- **FR20:** The system shall allow admins to view system logs and usage metrics (e.g., active rooms, error rates).

4. External Interface Requirements

4.1 User Interfaces

- **Applicant (Guest):** Clean registration forms, **Monaco Editor** for coding (with syntax highlighting and line numbers), and a console panel for output.
- **Administrator (Host):** Dashboard for session management, participant list with kick controls, and language toggle switches.
- **Design Principles:** Dark theme support, responsive layout, and clear error messaging (e.g., "Compilation Failed").

4.2 Hardware Interfaces

- **Client:** Desktop/Laptop with keyboard/mouse recommended; minimal screen resolution of 1366x768.
- **Server:** Standard cloud infrastructure (AWS/DigitalOcean) with sufficient compute for Docker containers.

4.3 Software Interfaces

- **Database: MongoDB** (v4.0+) using **Mongoose** ODM.
- **Web Server: Node.js** (v14+) with **Express.js**.
- **Real-time Engine: Socket.IO** (v4.x) for bidirectional event handling.
- **Execution Engine: Judge0 API** for sandboxed code running.

4.4 Communication Interfaces

- **HTTP/HTTPS:** For RESTful operations (Auth, Session CRUD).
- **WebSockets (WSS):** For persistent, low-latency editor synchronization and chat.
- **Data Format:** All data transmission shall be serialized as **JSON**.

5. Non-Functional Requirements

5.1 Performance

- **NFR1:** Support 500+ concurrent users without degradation.

- **NFR2:** Load core editor interface within 2-3 seconds.
- **NFR3:** Simple code execution must return output within 5 seconds; complex scripts capped at 10 seconds.
- **Latency:** Synchronization delay between clients should not exceed 200ms.

5.2 Security

- **NFR4:** Enforce HTTPS (TLS 1.2+) for all communications.
- **NFR5:** Encrypt sensitive data (passwords) using bcrypt/Argon2.
- **NFR6:** Strict RBAC to prevent Guest privilege escalation.
- **NFR7:** Mitigation of SQL/NoSQL injection and XSS attacks via input sanitization.
- **Sandboxing:** Execution environment must be isolated (Docker) with no network access and strict resource limits (cgroups).

5.3 Availability & Reliability

- **NFR8:** Target 99.9% uptime.
- **NFR9:** Automated daily database backups.
- **NFR10:** Auto-reconnection logic for WebSocket disconnects to prevent data loss.

5.4 Scalability

- **NFR11:** Horizontal scaling capability using Redis for WebSocket session management across multiple server instances.
- **NFR12:** Modular design allowing easy addition of new language runtimes.

5.5 Usability

- **NFR13:** Intuitive, menu-driven interface following Material Design/Fluent UI standards.
- **NFR14:** Syntax highlighting support for JS and Python.
- **NFR15:** Accessibility compliance (keyboard navigation).

5.6 Maintainability

- **NFR16:** Modular codebase separating Execution, User, and Collaboration logic.
- **NFR17:** Comprehensive developer documentation (README, API specs).

6. System Architecture Overview

6.1 Presentation Layer

- **Tech:** React.js.
- **Components:** Monaco Editor wrapper, Console display, Socket client.
- **Role:** Handles user interaction, captures keystrokes, and renders real-time updates.

6.2 Business Logic Layer

- **Tech:** Node.js / Express.js.
- **Role:** REST API for auth/session management; WebSocket Controller for room synchronization and event broadcasting; Execution Orchestrator for dispatching jobs to Judge0.

6.3 Data Layer

- **Tech:** MongoDB.
- **Stores:** User profiles, Session metadata, Code snapshots.
- **Caching:** Redis for active socket IDs and pub/sub messaging.

6.4 Integration Layer

- **Judge0 API:** External service receiving source code payloads, managing Docker containers for execution, and returning stdout/stderr.

7. System and Data Models

7.1 UML Diagrams (Description)

- **Use Case:** Actors (Applicant, Admin, System) interacting via Register, Login, Create Session, Edit Code, Run Code.
- **Sequence:** Client -> Server -> Judge0 -> Docker -> Judge0 -> Server -> Client (for execution flow).

7.2 ER Diagrams

- **User:** Stores credentials (username, email, password_hash).
- **Session:** Stores room state (room_id, host_id, language, is_active).
- **Snapshot:** Stores saved code versions (session_id, code_content, timestamp).

7.3 Workflow Diagrams

- **Session Flow:** Login -> Create Room -> Generate ID -> Redirect to Editor -> Connect Socket.
- **Execution Flow:** Run Clicked -> Code sent to API -> Sandbox Execution -> Result Returned -> Console Updated.

8. Validation and Acceptance Criteria

The system shall be validated against the following criteria:
- **Functional Testing:** Verification of all FRs (e.g., distinct users in a room see each other's edits instantly).
- **Integration Testing:** Successful data flow between React, Node, MongoDB, and Judge0.
- **Security Testing:** Penetration testing to ensure the sandbox cannot be escaped and unauthorized users cannot join rooms.
- **User Acceptance Testing (UAT):** Confirmation that a non-technical user can host a

session and run a "Hello World" program in Python/JS without errors.
- **Performance Testing:** System stability under a load of simulated concurrent connections.

9. Appendices

Appendix-A: Glossary

- **Applicant:** Guest Developer/Candidate.
- **Administrator:** Session Host/Interviewer.
- **Judge0:** The execution engine/verification authority.
- **MERN:** MongoDB, Express, React, Node.js.
- **JWT:** JSON Web Token.
- **CRDT/OT:** Algorithms for real-time text synchronization.
- **Docker:** Containerization platform used for sandboxing.

Appendix-B: Compliance Checklist

- [ ] Data security (Encryption/Hashing).
- [ ] GDPR Compliance (User data management).
- [ ] Audit Logging (Execution history).
- [ ] Sandboxing (Resource limits/Network isolation).