1. System Architecture

The system follows a layered architecture design to ensure separation of concerns, scalability, and maintainability. The architecture is divided into four primary layers: Presentation, Business Logic, Data, and Integration.

1.1 Presentation Layer

This layer handles the user interface and user interaction for both Applicants (Guests) and Administrators (Hosts).

- **Technology:** React.js.

- **Key Components:**
  - **Monaco Editor Component:** Wraps the Microsoft Monaco editor to provide the core coding experience with syntax highlighting and line numbers.
  - **Console Component:** Displays the standard output (stdout) and error logs (stderr) received from the execution engine.
  - **Socket Client:** Manages the persistent socket.io-client connection to the server for real-time updates.

- **Functionality:**
  - Captures user interactions such as login, form submission, and code typing.
  - Emits keystrokes and cursor movements as WebSocket events to the server.
  - Renders real-time updates from other users to ensure a synchronized view.

1.2 Business Logic Layer

This layer contains the core application logic, acting as the bridge between the client and the data/execution services.

- **Technology:** Node.js with Express.js runtime environment.

- **Key Modules:**
  - **REST API Controller:** Handles standard HTTP transactions for User Authentication (JWT), Session CRUD operations, and saving code snapshots.
  - **WebSocket Controller:** Manages the real-time "Rooms." It listens for events such as join-room, leave-room, and code-change. It uses broadcasting or synchronization algorithms (like Yjs/CRDTs) to ensure all clients in a session maintain the same state.
  - **Execution Orchestrator:** Validates "Run" requests against user quotas and dispatches the code to the Integration Layer.

1.3 Data Layer

This layer is responsible for the persistent storage and retrieval of application data.

- **Technology:** MongoDB (Primary Database) and Redis (Cache).
- **Data Stores:**
  - **Applicant Details:** Stores user profiles and hashed passwords.
  - **Application Records:** Stores session metadata, including Room IDs, Host IDs, and timestamps.
  - **Passport Information:** Stores saved code snapshots and execution history.
- **Caching Strategy:** Redis is used to store transient data, such as the list of active socket IDs in a specific room, and to enable Pub/Sub messaging between server instances for horizontal scaling.

1.4 Integration Layer

This layer facilitates secure communication with external systems and services, specifically for code execution.

- **Execution Engine (Judge0):**
  - Acts as the "Verification Authority" or "Police".
  - Receives code via HTTP POST requests.
  - Manages ephemeral Docker containers to sandbox the execution environment.
  - Compiles the code, runs it against standard inputs, and returns the output or error status.
- **Email Service:** External services (e.g., Nodemailer/SendGrid) used for sending transactional emails like password resets.

2. System and Data Models

2.1 Data Flow and Sequence

- **Code Execution Flow:**

  Client -> (POST /execute) -> Server -> (POST /submissions) -> Judge0 -> (Run in Docker) -> Judge0 -> Server -> (WebSocket emit) -> Client.

- **Collaboration Flow:**

  User A -> (Type 'a') -> Client A -> (Emit 'code-change') -> Server -> (Broadcast 'code-change') -> Client B -> (Update Editor).

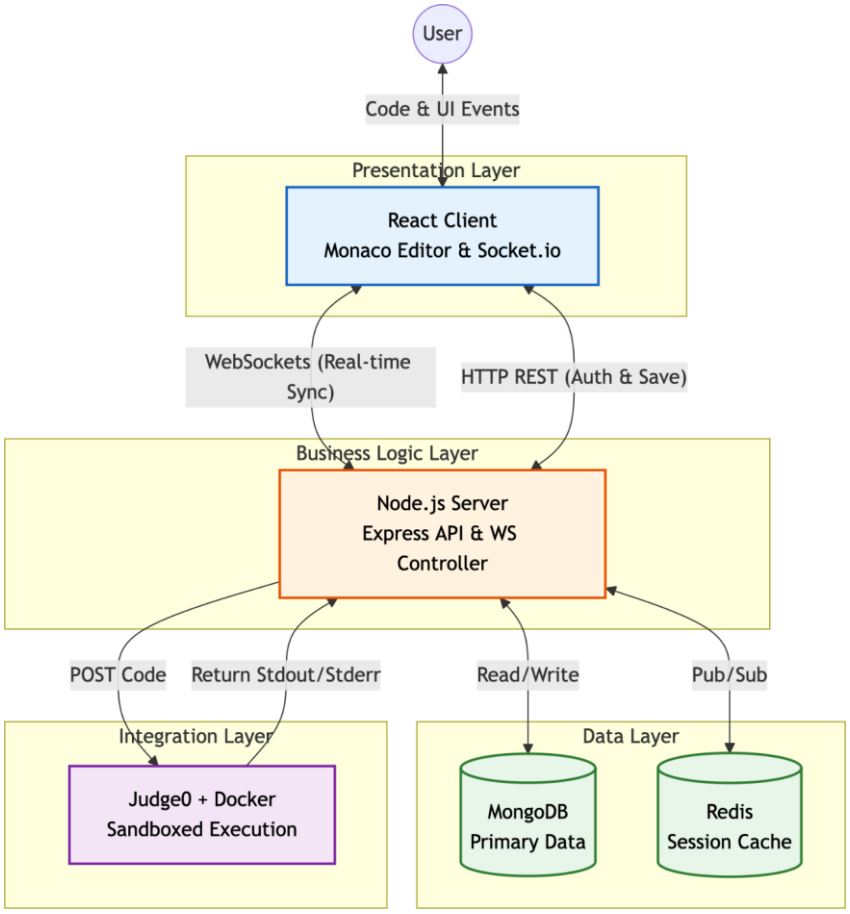
2.2 Database Schema (Entity-Relationship)

The system uses Mongoose for Object Data Modeling (ODM). The primary collections are:

- **User Collection:**
  - Fields: _id (ObjectId), username (String), email (String), password (Bcrypt Hash), createdAt (Date).

- **Session Collection:**
  - Fields: _id (ObjectId), roomId (UUID), host (Reference to User), language (String: 'javascript'/'python'), isActive (Boolean).

- **Verification (Snapshot) Collection:**
  - Fields: _id (ObjectId), session (Reference to Session), code (String), savedAt (Date).



2. Number of Modules

2.1. Functional Modules (from Use Case & Data Flow)

Based on the **Data Flow and Sequence** and your previous **Use Case** requirements, the system is composed of these primary modules:

- **Authentication Module:** Handles User Registration and Login.
- **Editor & Collaboration Module:** Manages real-time code changes using WebSockets and broadcasts them between users (User A to User B).
- **Code Execution Module:** Manages the flow from the Client through the Server to Judge0 and Docker for sandboxed execution.
- **Session Management Module:** Handles the creation and tracking of active coding rooms and language selection.
- **Data Persistence Module:** Manages the Mongoose collections for Users, Sessions, and Verification Snapshots.

2.2. Architectural Modules (from Component Diagram)

Your **Component Diagram** also groups the system into four major high-level modules:

- **Client Side:** React Frontend and Monaco Editor.
- **Server Side:** Node.js/Express API, Socket.IO Controller, and Execution Orchestrator.
- **Data Persistence:** MongoDB (Database) and Redis Cache.
- **Sandboxed Environment:** Judge0 API and Docker Containers.

3. System Requirements Specification: Real-time Collaborative Online IDE

Project: Real-time Collaborative Online IDE (RCO-IDE)
Deployment Strategy: Cloud-Native (Oracle Cloud Infrastructure)

1. Hardware Requirements

The system utilizes a client-server architecture. The server-side hardware is provisioned virtually via Oracle Cloud Infrastructure (OCI) to ensure high availability and zero-friction maintenance.

1.1 Server-Side (Cloud Infirastructure)

- **Provider: Oracle Cloud Infrastructure (OCI)** - *Always Free Tier*.
- **Instance Type: VM.Standard.A1.Flex** (Ampere Altra ARM Processor).
    - *Rationale:* This specific instance offers up to 4 OCPUs and 24 GB RAM for free, which is significantly more powerful than standard "micro" tiers (like AWS t2.micro), making it ideal for running resource-heavy Docker containers.
- **Processor (CPU): 4 vCPUs** (ARM64 Architecture).
    - *Requirement:* Dedicated cores are required to handle concurrent code execution requests (Judge0) without blocking the main Node.js event loop.
- **Memory (RAM): 24 GB**.
    - *Requirement:* Generous RAM is strictly necessary to run multiple Docker containers simultaneously (API Server, MongoDB, Redis, and ephemeral Sandbox containers for user code).
- **Storage: 50 GB** Boot Volume (NVMe SSD).
    - *Requirement:* High-speed SSD storage is critical for the rapid creation and destruction of Docker containers during code execution.
- **Operating System: Canonical Ubuntu 22.04 LTS (ARM64)**.
    - *Note:* Chosen for its native support for Docker on ARM architectures and long-term security updates.

1.2 Client-Side (User)

- **Device:** Desktop, Laptop, or Tablet.
- **Processor:** Dual-core processor (Intel i3/Ryzen 3) or befler.
- **RAM:** Minimum **4 GB** (Recommended **8 GB**) to support the Monaco Editor and React rendering.
- **Display:** Minimum resolution of **1366x768** pixels.
- **Network:** Broadband internet connection (Latency < 300ms) for real-time socket communication.

2. Software Requirements

2.1 Technology Stack

- **Frontend:**
  - **Framework:** React.js (v18+).
  - **Editor:** Microsoft Monaco Editor (VS Code core).
  - **State Management:** Redux Toolkit.
- **Backend:**
  - **Runtime:** Node.js (v18 LTS).
  - **Framework:** Express.js.
  - **Real-time:** Socket.io (v4.x).
- **Database & Caching:**
  - **Primary DB:** MongoDB Atlas (Cloud) or Self-Hosted MongoDB on OCI.
  - **Cache:** Redis Stack (Self-Hosted on OCI for managing socket rooms).

2.2 Infirastructure & Deployment Tools

- **Containerization: Docker & Docker Compose**.
  - *Usage:* The entire backend stack (Node API, Judge0, Redis) is containerized for consistent deployment across environments.
- **Execution Engine: Judge0 (Self-Hosted)**.
  - *Configuration:* Deployed as a Docker service on the OCI instance. Configured to run untrusted code inside isolated containers to prevent server compromise.
- **Reverse Proxy: Nginx**.
  - *Usage:* Handles SSL/TLS termination (HTTPS) and forwards traffic to the correct Docker container (Node.js API vs. Judge0 API).
- **CI/CD: GitHub Actions** (Optional).
  - *Usage:* Automates the build and deploy process to the Oracle Cloud instance via SSH whenever code is pushed to the repository.