	Marwadi University **Faculty of Technology** **Department of Information and Communication Technology**	
Subject: Capstone Project (**01CT0715**)	**Aim**: System Design and Architecture - Intermediate Review	
Project Documentation	**Date:-** 25-09-2025	**Enrollment No:-** 92200133030 92310133007

System Design and Architecture

1. System Architecture Overview

- The CodeArena platform is designed using a modern, decoupled architecture to ensure robustness, maintainability, and scalability. The overall design pattern is a Client-Server model featuring a Single Page Application (SPA) frontend that communicates with a Monolithic Backend API. This backend, in turn, interacts with a relational database and a specialized, external Microservice for remote code execution.
- This approach allows for a clear separation of concerns: the frontend is solely responsible for the user interface and experience, the backend manages all business logic and data orchestration, and the code execution engine handles its single, compute-intensive task in a secure, isolated environment.

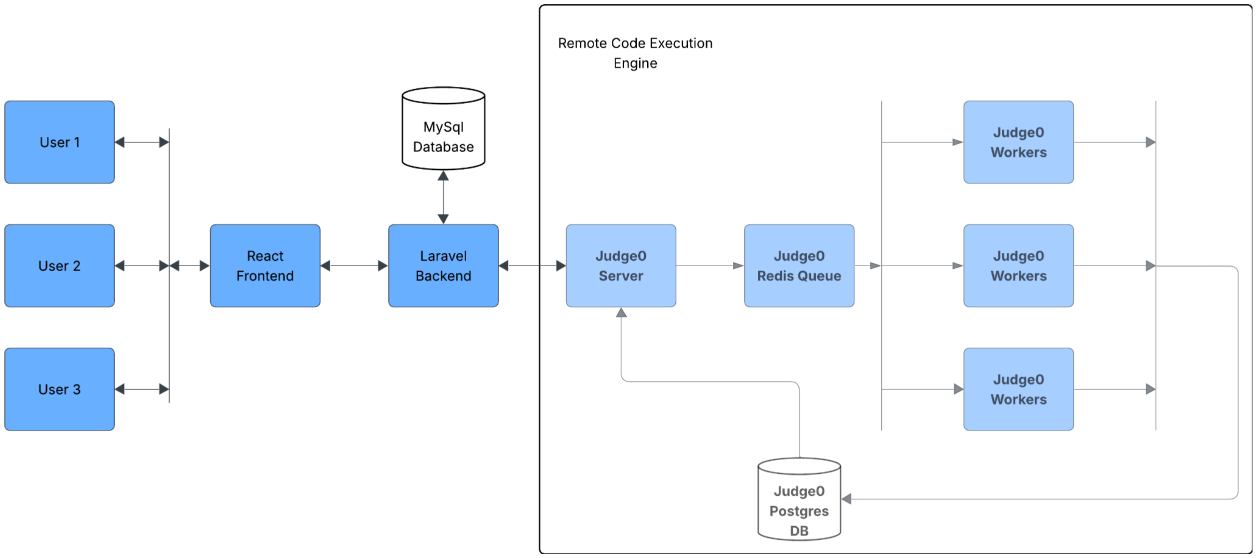
2. Modular Design

The system is broken down into four primary, independent modules. This modularity is key to enabling parallel development, simplifying maintenance, and allowing for future feature extensions without impacting the entire system.

2.1 Core Modules

1. **Frontend Application (Client):** This is the user-facing component built as an SPA. It is responsible for rendering all UI elements, managing user interactions, and communicating with the backend via a RESTful API. It contains no business logic.
2. **Backend Application (API Server):** The central nervous system of the platform. This module is responsible for user authentication and authorization, serving problem data, processing code submissions, managing user profiles, and interacting with the database. It exposes a set of secure API endpoints for the frontend.
3. **Database:** A relational database that serves as the single source of truth for the platform. It stores all persistent data, including user credentials, problem statements, test cases, and submission histories.
4. **Remote Code Execution (RCE) Engine:** A specialized microservice (Judge0) that is responsible for one task: securely compiling and running user-submitted code against predefined test cases and returning the execution results. It is completely isolated from the main application to ensure system security.

2.2 System Architecture Diagram



- Users interact with the system via a React Frontend, which serves as the user interface.
- The frontend communicates with a Laravel Backend, which handles application logic, authentication, and interacts with a MySQL Database for storing user and application data.
- The backend forwards code execution requests to the Judge0 Server, which manages the execution pipeline.
- Submissions are placed into the Judge0 Redis Queue for processing.
- Judge0 Workers pick up tasks from the queue, execute the code in isolated environments, and return results.
- Execution results and related data are stored in the Judge0 Postgres Database, and responses are sent back through the pipeline to the users. Once the Backend Receives the response from the Judge0 server, it also updates the local MySql Database.

This architecture ensures scalable, asynchronous, and reliable execution of code submissions while separating user management, application logic, and code evaluation components.

3. Technology Stack

Component	Technology	Justification
Frontend	**React.js**	React is the industry standard for building dynamic, responsive SPAs. Its component-based architecture enhances reusability. Axios is a robust library for handling API requests.
Backend	**Laravel (PHP)**	Laravel is a "batteries-included" framework with built-in security features, routing, and an expressive ORM (Eloquent) that simplifies database interactions and prevents common vulnerabilities like SQL injection.
Database	**MySQL**	MySQL is often preferred for its speed, ease of use, and wide community support, making it a reliable choice for managing academic data efficiently.
Remote Code Execution Engine	**Judge 0**	An open-source, engine that supports over 60 languages. Its use of isolated containers for code execution is critical for system security. Its simple REST API makes integration straightforward.
Infrastructure	**Docker**	Docker is used to containerize the Judge0 engine

4. Scalability Planning

The platform is designed to start lean but has a clear path for scaling as the number of users and submissions grows.

4.1 Identifying Potential Bottlenecks

1. **RCE Engine**: This is the most compute-intensive component. During peak hours (e.g., before an exam), a single Judge0 instance will become a major bottleneck, leading to long queue times for results.
2. **Database**: The submissions table will grow rapidly, potentially slowing down write operations and read queries for a user's submission history.
3. **Backend API Server**: A single server instance can only handle a finite number of concurrent HTTP requests.

4.2 Scaling Strategy

A multi-pronged strategy will be employed to address these bottlenecks.

- **Horizontal Scaling of the RCE Engine:** This is the highest priority. We will run **multiple, stateless Judge0 worker instances**. The Laravel backend will be configured to act as a load balancer, distributing batch submission requests across the pool of available Judge0 workers in a round-robin fashion. This is highly cost-effective and directly scales our code processing capacity.
- **Horizontal Scaling of the Backend:** When the API server becomes a bottleneck, we can place multiple instances of the Laravel application behind a **load balancer**. Since the backend is designed to be stateless (session state can be managed by a shared cache like Redis), this is a straightforward scaling path.