<table>
<tr><td></td><td colspan="2"><b>Marwadi University
Faculty of Technology
Department of Information and Communication Technology</b></td></tr>
<tr><td><b>Subject:</b> Capstone Project (<b>01CT0715</b>)</td><td colspan="2"><b>Aim</b>: Deployment and Operations</td></tr>
<tr><td><b>Project Documentation</b></td><td><b>Date:-</b> 25-09-2025</td><td><b>Enrollment No:-</b> 92200133030
92310133007</td></tr>
</table>

1. Deployment and Operations

This section details the transition of the CodeArena project from a development environment to a live, operational system. It covers the deployment process, the strategy for monitoring system health and performance, a comprehensive plan for ongoing maintenance, and challenges encountered during the implementation.

1.1 Deployment Process

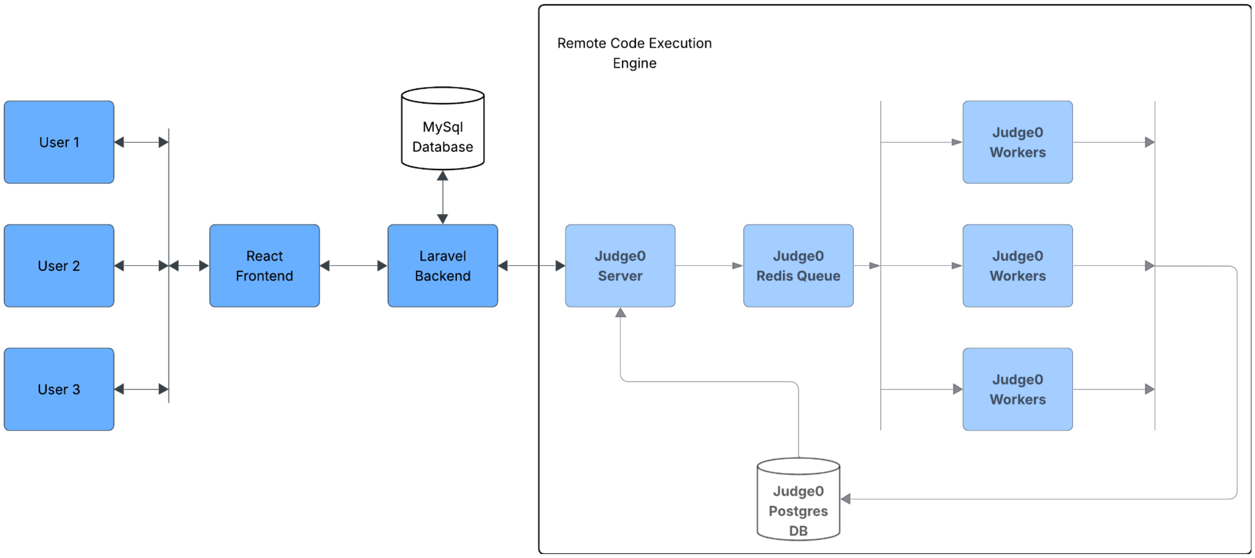
The deployment of CodeArena was executed in a phased approach, beginning with a controlled live deployment on the university's Local Area Network (LAN), with a clear roadmap for future public deployment on a university-provided Virtual Private Server (VPS).

1.2 Platform and Architecture

The current live environment is a dedicated server running **Ubuntu 22.04 LTS** with WSL. The Linux Subsystem handles the implementation of the Judge0 server, and Windows is used to host the backend and the frontend in the Local Area Network. This platform was selected for the initial rollout to facilitate a controlled test with a pilot group of users at no additional cost.

The entire Judge0 stack is containerized using **Docker and Docker Compose**, which ensures consistency and portability between environments. The architecture is orchestrated via a `docker-compose.yml` file that defines and links all necessary services for Judge0; it consists of the Judge0 Server, Database, Redis Queue, and the workers.

Then we have the Laravel Backend that will both work with the Judge0 service as well as our database. The Backend has endpoints that are set up to only allow incoming requests from the React deployment and block the rest by default due to the CORS policy.



1.3 Configuration and Launch

The deployment followed these steps:

1. **Server Provisioning:** The lab server was configured with Docker Engine and Docker Compose.
2. **Containerization:** The application was cloned from its Git repository, and the Docker images for the Judge0 were built.
3. **Configuration:** Environment variables, including database credentials and application secrets, were managed securely using a `.env` file on the host.
4. **Launch:** The entire stack was launched with the command `docker-compose up -d`.

1.4 Evidence of Live Deployment

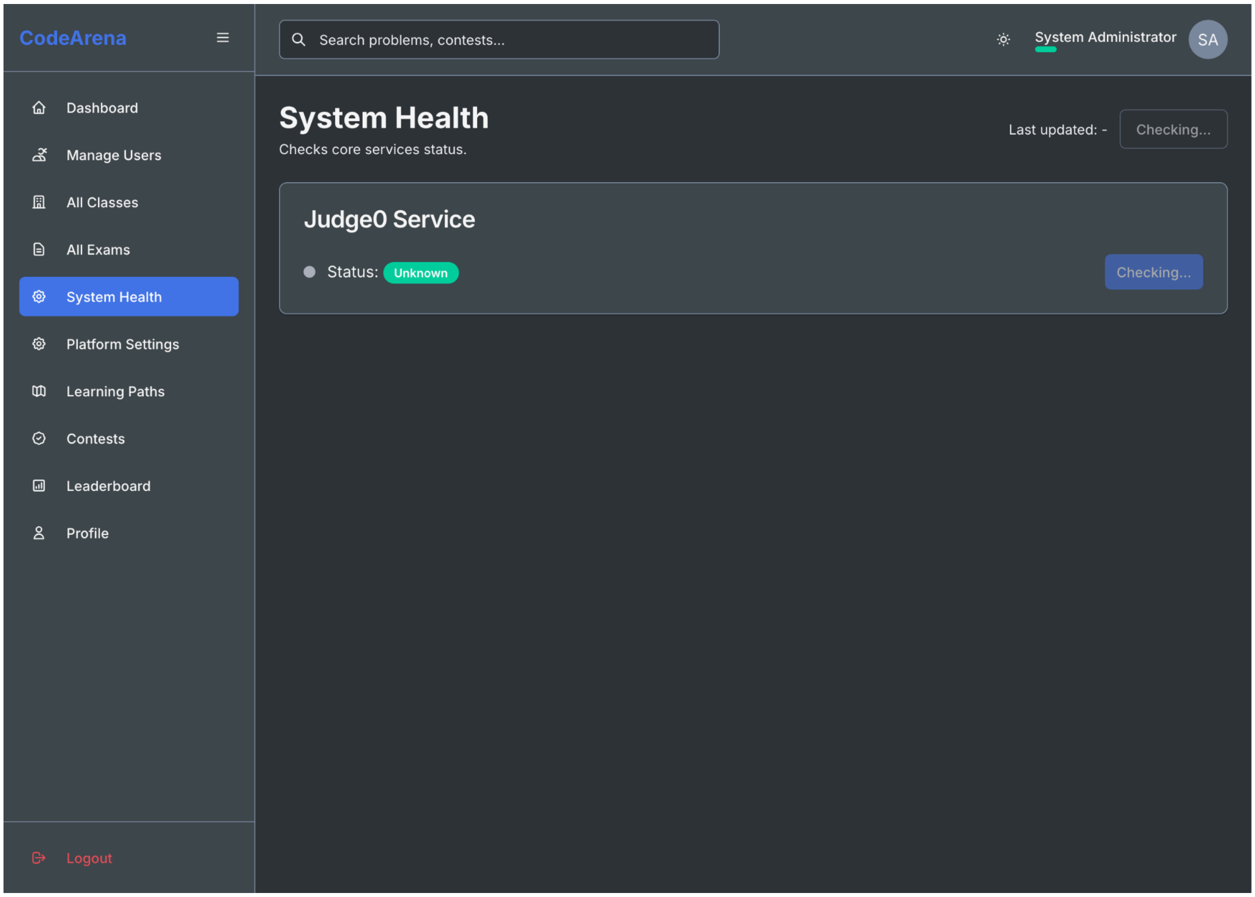
Semester-3 ICT Students trying out CodeArena Learning Path Module using Local Area Network

Semester-1 ICT Students trying out CodeArena Learning Path Module using Local Area Network

2. Monitoring Strategy

2.1 Monitoring Setup

Created a System Admin that is able to monitor and check for Judge0 Health Status.



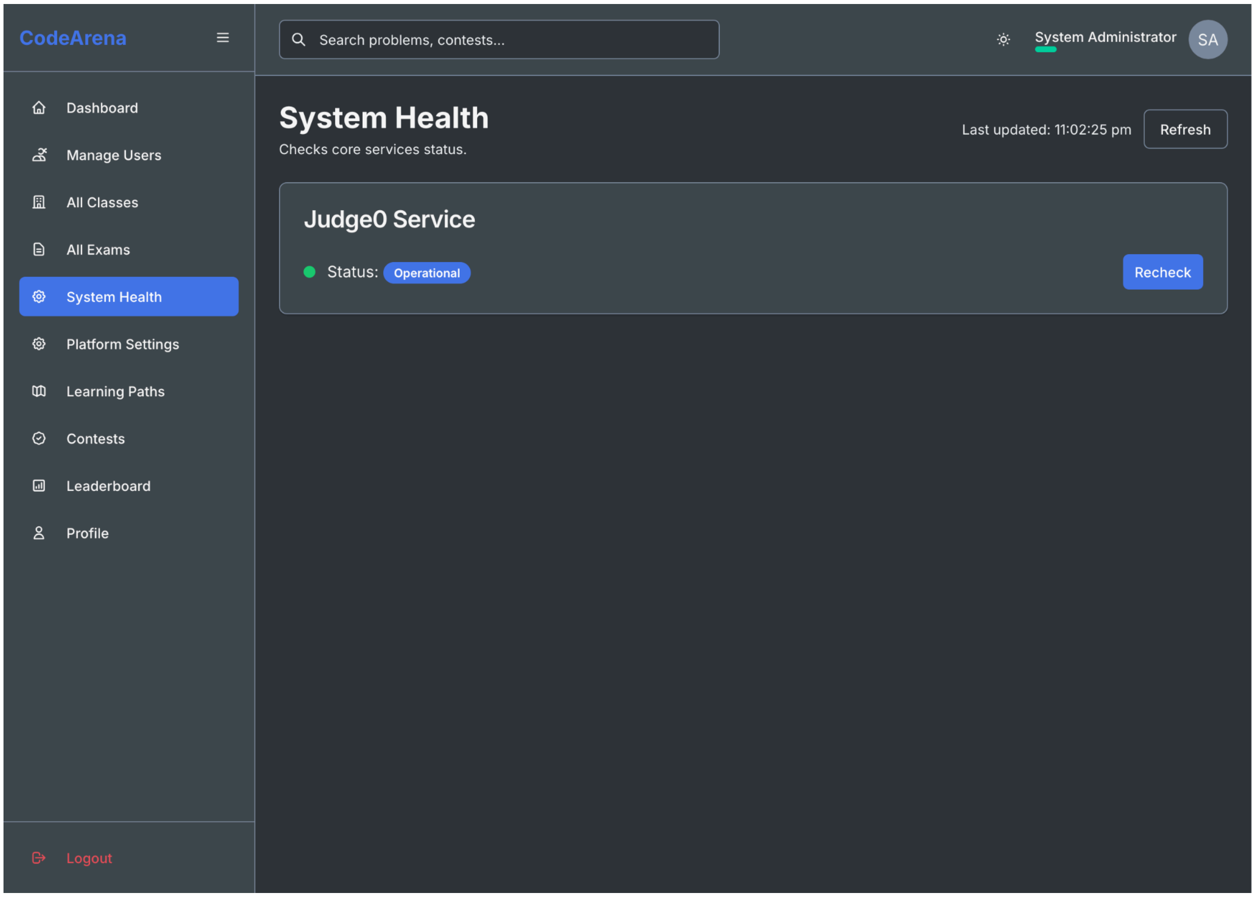
System Health

Checks core services status.

Last updated: -

Judge0 Service

● Status: Unknown



System Health

Checks core services status.

Last updated: 11:02:25 pm

Judge0 Service

● Status: Operational

2.2 Key Performance Indicators (KPIs)

Three core KPIs were identified to monitor the platform's operational performance:

1. API Response Time (95th Percentile): Measures the request-response loop for the backend API, ensuring a responsive UI. **Target: < 250ms**.
2. Code Submission Execution Time: Tracks the end-to-end duration for code judging, the platform's most critical function. Target: **Average < 5 seconds.**
3. Application Error Rate: The percentage of server-side requests resulting in 5xx errors, indicating application stability. **Target: < 0.2%.**

3. Schedule of Regular Maintenance Tasks

Frequency	**Task**	**Description**
Daily	Automated Database Backups	A cron job executes a script to perform a full backup of the MySQL database to a separate network drive.
Weekly	System Security Patches	The `apt` package manager is used to apply the latest security updates to the server's OS and software.
Monthly	Dependency Audit	`npm audit` and `composer outdated` are run to identify and plan updates for any vulnerable dependencies.

3.1 Updates and Bug Fixes

A structured workflow using Git is followed for all code changes. Bugs and feature requests are tracked via a ticketing system. All changes are developed on branches, tested manually, and then merged to the main branch for deployment, ensuring stability.

Mitigation of Potential Issues

- **Scalability Limits:** The architecture is designed for growth. The future VPS environment will allow for vertical scaling (adding CPU/RAM). If needed, horizontal scaling can be implemented by adding more application nodes behind a load balancer.
- **Hardware Failures:** The daily backup strategy is our primary mitigation against hardware failure. This ensures that the entire application and its data can be restored on a new server with minimal data loss.
- **Software Dependencies:** The monthly dependency audit proactively addresses risks from outdated or vulnerable libraries, a common challenge in modern software development.

4. Challenges

Several technical challenges were encountered and resolved during the deployment phase.

- **Cross-Origin Resource Sharing (CORS):** Initial communication between the React frontend and Laravel backend was blocked by browser security policies. Instead of implementing a permissive CORS policy on the backend, this was resolved at the infrastructure level.
- **High Resource Consumption by RCE Engine**: During stress testing, the Judge0 container consumed excessive CPU resources, which impacted the performance of other services.