High-Level Design: Online Judge System

Shashank Raman V

A2025B3

Django Team 2

**Introduction and Vision**

This Online Judge platform aims to offer a secure, scalable, and intelligent environment for coding challenges. Users will be able to browse problems, submit code in different languages, and receive feedback instantly, including AI-powered suggestions.

The mission is to not only provide automated grading and leaderboard features like traditional judges (e.g., LeetCode, HackerRank) but to enhance the user experience with AI that teaches, explains, and improves submitted code. Admin-users can also conduct contests and monitor the contest.

Target users include students, educators, and competitive programmers. This system can be used in academic settings or open developer communities.

Key highlights include:

* Secure and isolated code execution via Docker.
* Full-stack modern frontend using React.
* Django REST backend with JWT authentication.
* PostgreSQL for relational data with Redis caching.
* AI integration for feedback, hints, and similarity checking.
* Admins can create and schedule conte

**Frontend Architecture**

React (with TypeScript) will be used for an interactive frontend. The app will be styled with Tailwind CSS and feature reusable components.

Main Pages:

* Home: Navigation, overview, top problems.
* Problem List: Paginated/filterable list with difficulty tags.
* Problem Detail: Statement + code editor + sample test case.
* Leaderboard: Recent 10 submissions + filters.
* AI Panel: Optional section showing feedback/suggestions.
* Login: Signing in to access the platform.
* Register: New users can sign in.
* Admin page: Contest Management

Libraries & Tools:

* Axios for API requests.
* react-hook-form + yup for form handling.
* React Router for routing.

**Backend Architecture**

The backend will be built using Django with Django REST Framework (DRF). The app will be organized into modular Django apps, each handling a specific part of the logic.

Core Apps:

* accounts: Authentication and user management.
* problems: manages coding problems.
* testcases: stores test case data for each problem.
* solutions: handles code submissions and verdicts.
* judge: manages Docker execution and evaluation logic.
* contests: for timed competitions
* ai\_feedback: provide feedback and hints
* leaderboard: show rankings

Each app will expose APIs using DRF, secured using JWT tokens via djangorestframework-simplejwt. We'll ensure APIs follow REST best practices and are protected with permissions and rate limits.

**Database Design**

PostgreSQL is our main relational database. It offers great performance, indexing support, and data integrity.

Key Tables:

* User: Authenticated users (Django default).
* Problem: Title, statement, difficulty, etc.
* TestCase: One problem -> many test cases.
* Submission: User code, verdict, output, execution time.
* Feedback: AI-generated advice stored against each submission.
* ProblemScoring: Scoring for testcases.
* UserScoring: Score for users based on submissions.

Indexes will be added for performance-critical fields like created\_at and problem\_id.

Redis will be used for caching.

**Secure Execution with Docker**

Each submission will be executed in an isolated Docker container to prevent any security breaches.

Execution Pipeline:

1. User submits code.
2. API sends a task to a job queue.
3. Judge worker picks the task.
4. Worker launches a Docker container with the chosen language image.
5. Code is executed and compared with test case outputs.
6. Verdict and output are saved in the database.

Security is reinforced with container resource limits (CPU, RAM, time), seccomp profiles, and optional sandboxing using isolate. Only sanitized data is passed to prevent injections.

**AI Integration**

This is where our Online Judge shines. We'll integrate AI to improve feedback and engagement.

AI Capabilities:

* Code Analysis
* Optimization Suggestions
* Plagiarism Detection
* Hint Generation

Locally host models (e.g., CodeLlama) with libraries like transformers, langchain, or llama-cpp can be used. Feedback is asynchronously generated and stored, visible in the frontend as a side panel