A Layered architecture model for an online judge system provides a structured way to design the system, breaking it into distinct layers with specific responsibilities. Below the breakdown of a typical layered architecture for an online judge system:

1. Presentation Layer : (Frontend)

Purpose : Interact with users and provides a user-friendly interface.

Responsibilities:

- Display problem statements, user submissions, rankings, and contest details.
- Provide forms for code submission and user registration.
- Support features like live contest updates, notifications and feedback mechanisms.

Technologies:

- HTML/CSS/JavaScript (React, Angular or Vue.js)
- RESTful APIs or GraphQL for communication with the backend.

2. Application Layer (Backend)

Purpose: Handle business logic & coordinate between the frontend and other layers.

Responsibilities:

- Manage user authentication and session handling.
- Process submissions and pass them to the executio Layer
- Handle contest and problem management.
- Retrieve data from the database layer and present it to the frontend
- Enforce system rules (time limits, plagiarism checks

Technologies:

- Web frontworks like Node.js, Django, Flask or Spring Boot
- APIs for communication with execution and storage layers.

3. Execution Layer (Code Evaluation)

Purpose: Compile and execute submitted code in a secure, isolated environment.

Responsibilities:

- Compile that user's code in multiple programming language.
- Execute the code with test cases while enforcing time and memory limits.
- Capture output and compare it with expected results.
- Handle errors gracefully, such as runtime errors, compilation errors and timeouts

Technologies:

- Compiler/interpreter tools (GCC, Python, Java)
- Tools like Docker, Firejail or custom sendbox environments.

4. Data Layer (storage)

Purpose: Store and manage presistent data.

Responsibilities:

- Store user data (profiles, submissions, rankings)
- Store problem data (problem statements, test cases, constraints)
- Maintain logs of submissions, errors, and execution reports.

Technologies:

- Relational databases (MySQL, PostgreSQL) for structured data.
- NoSQL databases (MongoDB, Redis) for caching & real-time data.
- File storage for large binary data like problem test cases.