CodeArena – High Level Design (HLD)

(Version 1.0 & Future Vision)

Objective

Build an **Online Judge** platform (**CodeArena**) where users can:

- Create accounts and log in securely
- Browse and attempt coding problems
- Write and run C++ code in an online editor
- Test and submit their solutions
- View real-time compilation and execution results

Future versions (V2+) will extend to:

- Multiple languages (Python, JavaScript, etc.)
- Distributed queue processing
- CI/CD automation
- Advanced community and leaderboard features

Architecture Overview (V1)

Architecture Style:
Modular Monolith (scalable to microservices later)

Tech Stack (V1)

Layer	Tech
Frontend	React + TypeScript + TailwindCSS
Backend	Node.js + Express + TypeScript
Database	PostgreSQL
Code Execution	Docker-based sandbox (C++ only)

Auth	JWT (Access + Refresh Tokens)
Deployment	PM2 / Docker containers
Hosting	Any VPS
Editor	Monaco Editor

Core Modules (V1)

User Service

Handles all authentication and user management.

Responsibilities:

- Register / Login
- JWT token issuance and refresh
- Store basic user info & stats

Table: `users`

Column	Type	Description
id	UUID	Primary Key
username	VARCHAR	Unique
email	VARCHAR	Unique
password_hash	TEXT	Hashed via bcrypt
created_at	TIMESTAMP	—
updated_at	TIMESTAMP	—

Problem Service

Stores programming problems and their associated metadata.

Responsibilities:

- CRUD for problems (admin)
- Public listing with filters (difficulty, tags)
- Securely store sample and private test cases locally (in V1)

Tables:

`problems`

Column	Type	Description
id	UUID	Primary Key
title	VARCHAR	Problem name
description	TEXT	Full statement
difficulty	ENUM	`easy`, `medium`, `hard`
input_format	TEXT	—
output_format	TEXT	—
constraints	TEXT	—
sample_input	TEXT	Public example input
sample_output	TEXT	Public example output
created_at	TIMESTAMP	—

`test_cases`

Column	Type	Description
id	UUID	Primary Key
problem_id	UUID	FK to problems
input	TEXT	Input data
expected_output	TEXT	Expected output

is_public	BOOLEAN	Used for “Run Test” vs “Submit”

Storage (V1):
→ Test cases stored on a local **file system** or as text blobs in DB.

Code Execution Service

Responsible for:

- Compiling and executing user code inside a **sandboxed Docker container**.
- Running sample tests (“Run Test”) or full tests (“Submit”).
- Returning output, execution time, and compilation errors.

Supported language (V1): `C++ (g++)`

Execution Flow:

1. User clicks **Run Test** or **Submit**
2. Backend API receives request `{ code, problemId, mode }`
3. Backend:
    - Fetches corresponding test cases
    - Creates a temporary workspace
    - Spins up a **Docker container** with C++ compiler
    - Writes user code + input files
    - Compiles (`g++`)
    - Executes with input redirection
    - Captures stdout/stderr
    - Cleans up container
4. Returns structured result JSON

Sample Response:

```
{
  "status": "success",
  "compileError": null,
  "testResults": [
    { "case": 1, "status": "Passed", "time": "0.12s" },
    { "case": 2, "status": "Failed", "expected": "42", "got": "24" }
  ]
}
```

Submissions & Results

Stores every user attempt for audit and history.

Table: `submissions`

Column	Type	Description
id	UUID	Primary Key
user_id	UUID	FK to users
problem_id	UUID	FK to problems
code	TEXT	Source code
language	VARCHAR	‘cpp’
mode	ENUM	‘test’ or ‘submit’
status	ENUM	‘success’, ‘compile_error’, ‘runtime_error’
result_json	JSONB	Test case results
runtime_ms	INT	—
created_at	TIMESTAMP	—

Frontend (V1)

Framework: React + TypeScript + TailwindCSS
Editor: Monaco Editor (used by VSCode)

Core Pages

Page	Description
`/login`	Login with email/password
`/signup`	Create account
`/problems`	List all problems
`/problem/:id`	Problem detail + code editor + results
`/profile`	User profile & submission history

Problem Details Page Layout

```
----------------------------------------------------
| Problem Title | Difficulty | Tags                |
|--------------------------------------------------|
| Left Panel: Problem statement                    |
| Right Panel: Code editor (Monaco)                |
|--------------------------------------------------|
| [Run Test] [Submit]                              |
|--------------------------------------------------|
| Results Section (compile or test output)         |
----------------------------------------------------
```

API Design (V1)

Auth APIs

```
POST /api/auth/signup
POST /api/auth/login
GET /api/auth/me
```

Problem APIs

```
GET /api/problems
GET /api/problems/:id
```

Code Execution APIs

```
POST /api/execute/test
POST /api/execute/submit
```

Submissions

```
GET /api/submissions
GET /api/submissions/:id
```

V2 Roadmap (Planned Enhancements)

Category	Feature	Description
Language Support	Add Python, JS, Java	Extend Docker runtime with multi-language templates
Queue System	Redis Queue + Worker	Asynchronous code execution for scalability
Sandbox	Separate Judge Service	Dedicated worker instances for parallel jobs
CI/CD	GitHub Actions	Automated test + deploy
Leaderboard	Rankings by score/time	Gamify platform
UI/UX	Dark mode, progress tracking	Better user experience

Security & Reliability

Concern	Implementation
Code Isolation	Docker sandbox per execution
Time Limit	Max 3s per test case
Memory Limit	Configurable per container
Network Access	Disabled in sandbox
Auth	JWT-based access & refresh tokens
Passwords	Bcrypt hashing
Rate Limiting	Basic IP-level throttling on `/execute`

Deployment Plan (V1)

Component	Deployment
Frontend	Netlify
Backend	VPS
Database	PostgreSQL
Code Runner	Docker container running on same VPS (for now)

Summary Flow (V1)

```
User → React Frontend → Express API → Docker Runner → PostgreSQL
```

1. User logs in → gets JWT
2. Views problems → picks one
3. Writes C++ code → clicks “Run Test”
4. API executes inside Docker → returns result JSON
5. User reviews output → submits final solution
6. Results stored in DB and displayed in profile