Contest Platform with Redis - System Design

1. System Goals

- Handle fast contest registrations (no DB bottleneck).
- Provide a real-time leaderboard.
- Save results permanently after the contest ends.
- Run on Windows host (Redis via WSL/Docker).

2. Architecture

- Frontend: Registration, submissions, real-time leaderboard (WebSockets or SSE).
- Backend (Node.js + Express): APIs for register, submit, leaderboard, finalize contest.
- Redis: In-memory store (Hashes, Sorted Sets, Sets).
- Database: Persistent storage (MySQL/Postgres/MongoDB).

3. Redis Schema Design

- User Details (per contest): contest:{contestId}:user:{userId} → Hash (username, email, score, etc.).
- Users in Contest: contest:{contestId}:users → Set of userIds.
- Leaderboard: contest:{contestId}:leaderboard → ZSET (userId → score).

4. Workflow

- Registration: HSET user details, SADD to users, ZADD to leaderboard with score=0.
- Score Update: ZINCRBY in leaderboard, HINCRBY in user hash.
- Leaderboard Fetch: ZREVRANGE leaderboard with scores.
- Contest End: SMEMBERS to get all users, HGETALL for details, store in DB, DEL contest keys.

5. Running Redis on Windows

- Option 1: WSL (Windows Subsystem for Linux) → Install Ubuntu → apt install redis-server.
- Option 2: Docker Desktop → docker run --name redis -p 6379:6379 -d redis.
- Option 3: Redis for Windows (unofficial build).
- Recommendation: Use WSL or Docker for easier management.