BrainBolt - Adaptive Infinite Quiz Platform

A. Problem Statement

Build an adaptive infinite quiz platform that serves one question at a time. Difficulty increases after correct answers and decreases after wrong answers.
You must design:

- an basic adaptive algorithm for next determining question based on current user score and streak,
- a user score/metrics model that reflects edge cases,
- a streak system where higher streak increases score on correct answers,
- live leaderboards for total score and current streak.

NOTE:

- Questions list with varying difficulty can be seeded as a static list.
- Please read the mandatory submission guidelines **here**

B. Product Requirements

Core Flow

- The system serves exactly one question at a time per user session.
- Correct answers increase difficulty within bounds.
- Wrong answers decrease difficulty within bounds.

Adaptive Algorithm Requirements

- Define how difficulty changes per answer.
- Explicitly address ping-pong instability: rapid oscillation between two difficulties (e.g., correct/wrong alternation causes difficulty to flip between 5 and 6 forever). You must propose a stabilizer, such as:
    - momentum or confidence score,
    - rolling window adjustment,

    - hysteresis band (small buffer before changing difficulty),
    - minimum streak required to increase difficulty.

Streaks

- Streak increments on correct answers and resets on incorrect.
- Streak multiplier affects score.
- Streak multiplier must be capped.

Scores/Metrics

- User score must incorporate:
    - difficulty weight,
    - accuracy,
    - streak multiplier,
    - [optional] recent performance window.

Live Leaderboards

- Two live leaderboards:
    - total user score,
    - current user streak.
- Leaderboards must update immediately after each answer and keep showing the current rank of the authenticated user.

Real-Time

Following must update in real time and be reflected in the very next question response:

- user score,
- current streak,
- max streak,
- adaptive difficulty,
- leaderboard rankings.

C. Expected Deliverables

Frontend Web App

- Build a SPA using Next.js + React (preferred) or another modern framework.
- Create a reusable component library (high evaluation weight): composable, accessible, scalable.
- Use design system tokens only (no hardcoded CSS values): colors, spacing, typography, radius, shadows, breakpoints.

- Add responsive design; light/dark mode preferred.
- Implement lazy loading / code splitting for non-critical routes/components (dynamic imports).
- Use SSR for at least one meaningful page/route; use CSR only where needed for interactivity.
- Ensure good performance (avoid unnecessary re-renders, efficient lists, memoization where appropriate).
- Use TypeScript preferred; include ESLint/Prettier and clean project structure.

Low-Level Design (LLD)

- Class/module responsibilities.
- API schemas (request/response).
- DB schema with indexes.
- Cache strategy (keys, TTL, invalidation).
- Pseudocode for adaptive algorithm and score calculation.
- Leaderboard update strategy.

Edge Cases

- Explicit list of adaptive edge cases and how scoring handles them.
- Include ping-pong and boundary conditions.

Suggested API Design

```
GET /v1/quiz/next
Request: userId, optional sessionId
Response: questionId, difficulty, prompt, choices, sessionId,
stateVersion, currentScore, currentStreak

POST /v1/quiz/answer
Request: userId, sessionId, questionId, answer, stateVersion,
answerIdempotencyKey
Response: correct, newDifficulty, newStreak, scoreDelta, totalScore,
stateVersion,
         leaderboardRankScore, leaderboardRankStreak

GET /v1/quiz/metrics
Response: currentDifficulty, streak, maxStreak, totalScore,
         accuracy, difficultyHistogram, recentPerformance
```

```
GET /v1/leaderboard/score
Response: top N users by total score

GET /v1/leaderboard/streak
Response: top N users by max streak
```

Suggested Data Model

```
users (id, createdAt)
questions (id, difficulty, prompt, choices, correctAnswerHash, tags)
user_state (userId, currentDifficulty, streak, maxStreak, totalScore,
lastQuestionId, lastAnswerAt, stateVersion)
answer_log (id, userId, questionId, difficulty, answer, correct,
scoreDelta, streakAtAnswer, answeredAt)
leaderboard_score (userId, totalScore, updatedAt)
leaderboard_streak (userId, maxStreak, updatedAt)
```

Caching Expectations

- Cache user state in Redis.
- Cache question pools per difficulty.
- Explain invalidation strategy and how real-time updates are guaranteed.
- Evaluate which metrics data can be cached.

Containerization + Resource Constraints

- Dockerized application + services (if any).
- Provide a Dockerfile and docker-compose (or equivalent).
- Single command to build and run the application you built.

Non-Functional Requirements

- Strong consistency for streak and score updates per user.
- Idempotent submitAnswer.
- Rate limiting to avoid abuse.
- Stateless app servers.

Few Edge Cases You Must Handle

- Streak reset on wrong answer
- Streak decay after inactivity.
- Duplicate answer submission should not update streak twice (idempotency).

D. Evaluation Rubric

- Frontend (25%)
- LLD rigour and correctness (25%)
- Functional Requirements (25%)
- Edge case handling (15%)
- Operational concerns and real-time correctness (10%)

IMPORTANT NOTE:

- Your submission for the assignment MUST be via a public Github repository.
- Submitted Github repository MUST have the following,
    - Single command to run your entire application stack (Backend, Frontend, Database etc)
    - Demo video in project root folder. This video must explain all the features implemented and show a working demo of the features and a walkthrough of both Frontend and Backend codebases.
- Failing to submit the above will lead to immediate disqualification.