CodingHub — Executive Abstract & Software Requirements Document

Executive Abstract

The CodingHub project is a full-stack, LeetCode-style learning and submission platform designed to help users practice programming problems, receive AI-powered code explanations, track daily streaks and acceptance rates, and manage submissions and community discussions. It primarily serves learners (students, interview candidates), content creators/admins, and moderators by providing an interactive React frontend, a Node/Express API, and a MongoDB-backed data layer.

The architecture comprises a React single-page application (SPA) deployed via GitHub Pages and a Node.js/Express backend exposing RESTful APIs for authentication, problem sets, submissions, streak tracking, profiles, discussions, and AI explanations via Google Gemini. Data persistence uses MongoDB with Mongoose models. The frontend communicates with backend routes under `/api/*`; the backend integrates external services (Gemini) and includes admin tooling for bulk question uploads and system initialization.

Software Requirements (SR) Document

1. Project Overview

- **Purpose:** Deliver a scalable coding practice platform with AI assistance, structured problem flows, submission tracking, and streak/acceptance analytics.
- **Scope:** Web-based SPA frontend (React) and REST API backend (Express/Mongoose) supporting users, admins, and moderators, with external AI integration.
- **Primary Features:** Problem browsing and detail views; code submission & history; AI code explanations; streak tracking; acceptance rate analytics; user profiles & settings; discussions; admin uploads and initialization.

2. System Goals & Objectives

- **Learning Efficiency:** Clear problem statements, test execution, and structured AI explanations.
- **Progress Tracking:** Submission history, acceptance rate analytics, streak tracking, charts.
- **Admin Productivity:** Upload/manage questions, configure settings, initialize admin.
- **Operational Quality:** Reliability, security, performance, maintainability, accessibility.

3. User Roles & Use Cases

- **Roles:**
  - **Learner/User:** Browse problems; open detail with tabs (Description/Submissions); submit solutions; review results and expected output; request AI explanations; track streaks and acceptance; participate in discussions; manage profile/settings.
  - **Admin:** Upload/manage questions (JSON formats per `ADMIN_UPLOAD_FORMAT.md`); moderate content; initialize system users (`utils/initAdmin`); manage practice/problem metadata and settings.
  - **Moderator (optional):** Review and moderate discussions; flag content; handle reports.

- **Key Use Cases:**
  - Browse/search/filter problems; open `ProblemDetail` view with tabbed interface.
  - Submit code; auto-switch to Submissions tab; view results, complexity, expected output.
  - Request AI code explanation via `/api/explain` (Gemini) with heuristic filter for non-code inputs.
  - Track daily streaks and acceptance rate; visualize progress (Recharts).
  - CRUD for discussions via `/api/discussions` with moderation workflows.
  - Edit profile and update settings (e.g., dark mode, notifications) via `/api/profile` and `/api/settings`.
  - Admin uploads question sets; validate/persist to MongoDB; manage categories/difficulty.

4. Functional Requirements

- **Authentication & Authorization:**
  - Register/login; JWT issuance and validation; role-based access for admin endpoints.
  - Profile retrieval/update via `/api/profile`; secure token handling.
- **Problems & Practice:**
  - List problems; get detail; include categories/difficulty/constraints.
  - Submit solutions; persist submissions with status, runtime, memory, timestamp via `/api` submission routes.
  - Calculate and store acceptance rates; expose history per user/problem.
- **AI Explain:**
  - POST `/api/explain` with `code` payload. If input is not code (heuristic), return onboarding message.
  - Call Gemini `v1beta` endpoint (`gemini-2.0-flash:generateContent`); return structured markdown sections: Explanation, Methods, Main Logic, Complexity, Tips, Expected Output.
- **Streaks:**
  - Endpoints under `/api/streak` for daily streak tracking; support uploads via JSON (`admin-upload-streak.json`, `streak-questions-5.json`).
- **Discussions:**
  - CRUD APIs under `/api/discussions` for threads, comments, and moderation.
- **Admin:**
  - `/api/admin` routes for question uploads, practice/problem management, system settings.
  - Initialization script to create admin user on startup.
- **Settings:**
  - Read/write user preferences via `/api/settings`; frontend dark mode and UI behavior.
- **Tutorials & Guides:**
  - `/api/tutorials` provides curated learning content and guides.
- **Health/Test API:**
  - `/api/test` for health checks and diagnostics.
- **Frontend Application:**
  - SPA with routing (`react-router-dom`) for Problems, Detail, Submissions, Profile, Discussions, Settings.
  - Markdown rendering (`react-markdown`), syntax highlighting (`react-syntax-highlighter`), Monaco editor (`@monaco-editor/react`) for code editing.
  - Notifications (`react-hot-toast`), iconography (Heroicons/Lucide/React Icons), charts (`recharts`).

5. Non-Functional Requirements

- **Security:**
  - JWT-based auth; bcrypt password hashing; role-based authorization for admin endpoints.
  - CORS policy enabled; restrict allowed origins in production; enforce HTTPS.
  - Input validation and rate-limiting on sensitive endpoints (e.g., AI explain).
- **Performance:**
  - Typical API response <300ms for non-AI requests.
  - Optimize Mongoose queries; add indexes for submissions/history; avoid N+1 queries.
- **Reliability:**
  - Graceful handling of AI service errors and timeouts; fallback messages.
  - MongoDB connection timeout set (`serverSelectionTimeoutMS = 5000`); retry or alert on failure.
- **Scalability:**
  - Stateless Node/Express suitable for horizontal scaling; MongoDB supports sharding.
- **Maintainability:**
  - Clear folder structure: `routes/`, `controllers/`, `models/`, `middleware/`, `utils/`.
  - Extensive documentation under `ReadmeFiles/` covering flows, visuals, quick starts.
- **Usability & Accessibility:**
  - Beginner-friendly explanations; consistent tabbed UI; keyboard accessibility per docs.
  - Responsive design; accessible color palette and contrast.

6. Tech Stack Description

- **Frontend:** React 19, React Router, Monaco Editor, React Markdown, React Syntax Highlighter, Recharts, React Hot Toast, Heroicons/Lucide/React Icons; built with `react-scripts` (CRA 5); deploy via `gh-pages` to GitHub Pages (`homepage` configured).
- **Backend:** Node.js, Express 5, Mongoose 8, JWT, bcryptjs, dotenv, cors, axios; dev tooling with `nodemon`.
- **Integrations:** Google Generative AI via `@google/generative-ai` and HTTP POST to Gemini `v1beta` (`gemini-2.0-flash`).
- **Data:** MongoDB with Mongoose models for users, problems, submissions, discussions, streaks, profiles, and settings.

7. System Architecture

- **API Flow:**
  - Client → `/api/*` endpoints (auth, explain, practice, questions, submissions, streak, profile, settings, discussions).
  - AI explain: client → `/api/explain` → Gemini POST → normalize response → client renders markdown explanation.
  - Admin upload: admin → `/api/admin` → validate payload → persist to MongoDB → indexes update.
- **Database Flow:**
  - Entities persisted via Mongoose models: `User`, `Problem`, `Submission`, `Discussion`, `Streak`, `Profile`, `Settings`.
  - Submission lifecycle: store status/result; derive acceptance rate from outcomes; maintain history for analytics.
- **Frontend–Backend Interaction:**

- React SPA consumes REST endpoints via `axios`; renders markdown/code, charts, and submission history.
- Auth tokens stored client-side; attached to protected requests.
- UI patterns documented in `ReadmeFiles` (tabs, back-button, responsive layouts).

8. Integration Details

- **External APIs:**
  - Google Gemini: `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key={GEMINI_API_KEY}`.
  - Heuristic filter avoids non-code prompts to reduce cost and improve UX.
- **Hosting/Deployment:**
  - Frontend via GitHub Pages using `gh-pages` script.
  - Backend as Node/Express service; environment managed via `.env`.
- **Configuration:**
  - `.env` requires `MONGODB_URI`, `GEMINI_API_KEY`, `PORT`, JWT secret(s).
  - CORS currently permissive (`*`) for development; restrict in production.

9. Constraints & Assumptions

- **Constraints:**
  - Dependence on external AI availability and quotas; handle rate limits/outages.
  - MongoDB must be reachable; startup guarded with shorter server selection timeout.
  - CRA configuration limitations without eject; advanced customizations may require migration.
- **Assumptions:**
  - Users are on modern browsers; mobile responsiveness implemented per docs.
  - Problem and submission schemas align with documented formats (`LEETCODE_SUBMISSION_SYSTEM.md`, `ADMIN_UPLOAD_FORMAT.md`).
  - JWT secrets configured; HTTPS enforced in production.
  - Discussion moderation policies to be finalized by admins.

Appendix

- **Key Files & Folders:**
  - Frontend: `client/` (SPA, assets, `package.json` with dependencies and `homepage`).
  - Backend: `server/` (`index.js`, `routes/`, `controllers/`, `models/`, `utils/`).
  - Documents: `ReadmeFiles/` (Quick Start, Complete Guide, Visual Guide, Acceptance Rate, Code Reference, Implementation Summary).
- **Representative Backend Routes:**
  - `/api/auth`, `/api/explain`, `/api/chat-history`, `/api/tutorials`, `/api/practice`, `/api/questions`, `/api/test`, `/api/admin`, `/api/streak`, `/api` (submissions), `/api/discussions`, `/api/profile`, `/api/settings`.
- **Environment Variables:**
  - `MONGODB_URI`, `GEMINI_API_KEY`, `PORT`, `JWT_SECRET` (and variants), CORS settings.