Reborne.fun API Overview

1. Project Architecture

• We’re running a Next.js 15 monorepo that serves both the UI and a minimal API via React Server Components and client-side Firebase SDK calls. There is no separate backend server: data reads and writes go directly from the browser to Firestore using our client SDK wrappers.

• No Dedicated REST/GraphQL Layer (yet). All data access is encapsulated in utility functions:
- getRevenue(), addRevenue()
- getLeaderboard(), upsertLeaderboardEntry()
- getRevivedRanking(), incrementRevivalCount()
- (Optional Firebase Cloud Functions for scheduled distribution)

2. Available Data Endpoints

GET /api/revenue → getRevenue() → Top-20 distribution entries (rank, address, rewards)
POST /api/revenue → addRevenue(address, amount) → Increment user rewards
GET /api/leaderboard → getLeaderboard() → Top leads (wallet, token, marketCap, SOL earned, status)
POST /api/leaderboard → upsertLeaderboardEntry() → Upsert a lead’s stats
GET /api/revived → getRevivedRanking() → Most-revived tokens (id, name, count, marketCap)
POST /api/revived → incrementRevivalCount() → Increment token Revival count

3. Next Steps & LLM Overview

1) Spin Up Thin API Adapter:
   • Add routes under src/pages/api/... or a small Express/Next.js layer that calls our existing functions and returns JSON.

2) Document & Summarize Endpoints:
   • Generate an OpenAPI spec or Markdown doc listing each route, parameters, and schemas.

3) LLM-Powered Summary:
   • Run an LLM over the spec to produce a human-friendly overview your Telegram bot can consume.

```
Example: GET /api/revenue
  Response 200:
    [
      {
        "rank": 1,
        "address": "5F3sa2...",
        "amount": 1000000,
        "rewards": 0.25
      },
      …
    ]
```