ChatGPT

Do deep research on kmages/Gemin-Eye and produce an annotated improvement plan for an agentic Replit coder

Executive summary

Gemin-Eye is a Replit-oriented, Node.js 20 / Postgres app that monitors sources (Reddit RSS, Google Alerts RSS, plus "Spy Glass" bookmarklets) and uses Gemini to score leads and generate responses, then delivers them to Telegram. The repo already includes a landing page, dashboard, onboarding/admin routes, monitors, and a Telegram bot/webhook workflow. [1]

The highest-leverage improvements (ROI × risk reduction) cluster into four themes:

- **Make onboarding "frictionless" and unambiguous**: the README and `.env.example` exist but are effectively minified into a few long lines, and the `.env.example` is not in a standard multi-line `.env` format, which increases setup failure probability for both Replit and non-Replit users. [2]
- **Fix trust-boundary correctness immediately**: `escapeHtml()` in `server/utils/html.ts` is syntactically/semantically broken, while Telegram messages are explicitly sent using `parse_mode: "HTML"`. This combination is high risk (format breakage + possible injection-like issues). [3]
- **Make AI JSON parsing truly type-safe and reusable**: `parseAIJsonWithRetry` and `safeParseJsonFromAI` exist, but the parsing approach is brittle (single regex capture of `{…}`) and the TypeScript signatures are underspecified. Multiple flows also bypass schema validation (e.g., scan endpoints/Telegram analysis). [4]
- **Production hardening for autoscale**: Replit deployment is configured for autoscale; your rate limiter is purely in-memory. That means rate limits won't hold across instances or restarts—high risk for abuse/cost control as you sell this as a SaaS. [5]

The plan below is optimized for an **agentic Replit coder**: small PRs, explicit file targets, concrete diffs, and "what to run" commands on Replit.

Repository snapshot and Replit runbook

What the repo expects at runtime

- Replit config shows **Node 20**, **PostgreSQL 16**, and `npm run dev` as the run command; deployment target is **autoscale**, and production sets `MONITORING_DISABLED="true"` by default. [6]
- Server startup enforces required env vars `DATABASE_URL` and `SESSION_SECRET` and warns if `TELEGRAM_BOT_TOKEN` / `AI_INTEGRATIONS_GEMINI_API_KEY` are missing (some features disabled). [7]

- Database is Drizzle over `pg.Pool` bound to `DATABASE_URL`; schema is in `shared/schema.ts`; migrations output to `./migrations`. [8]

Replit commands for the agent (procedure)

In a fresh Replit shell (or locally), the agent should run:

```
# Clone
git clone https://github.com/kmages/Gemin-Eye.git
cd Gemin-Eye

# Install (CI-compatible)
npm ci

# Create env (then fill in via Replit Secrets UI or editing .env)
cp .env.example .env

# Provision DB schema
npm run db:push

# Dev server
npm run dev
```

This matches the README's "Getting Started" flow (though the README itself needs formatting cleanup). [9]

Agent workflow note: your `replit.md` explicitly instructs "Always push code to GitHub after every commit" (assuming GitHub remote is connected), and `.replit` includes a GitHub agent integration. [10]

Priority fixes and recommended changes

Below are the recommended changes prioritized by ROI and risk, packaged into numbered PRs. For each change: description, why, exactly where, example diff, verification steps, effort/PR sizing, and primary source links (raw URLs in code blocks as requested).

PR 1: Onboarding polish for humans and agents

Concise description
Rewrite/format `README.md` and `.env.example` into standard multi-line formats; communicate "Replit-first" + "local dev" clearly (Node 20). Add a "Replit agent" checklist section.

Why it matters (impact/risk)
This reduces install failures and support burden. Right now, `README.md` is effectively a few extremely long lines, and `.env.example` is a single long line of key/value/comment fragments—both increase setup error likelihood. [2]

Exact files/paths to change - `README.md` [11]
- `.env.example` [12]
- (Optional, doc alignment) `replit.md` [13]

Small diff example

```
diff --git a/.env.example b/.env.example
--- a/.env.example
+++ b/.env.example
@@
-# Database (required) DATABASE_URL=postgresql://user:password@host:5432/dbname
# Session (required) SESSION_SECRET=your-session-secret-here ...
+# Database (required)
+DATABASE_URL=postgresql://user:password@host:5432/dbname
+
+# Sessions / Auth (required)
+SESSION_SECRET=replace-me-with-a-long-random-secret
+
+# Replit OIDC (Replit auto-sets many of these; local dev may require explicit
values)
+ISSUER_URL=https://replit.com/oidc
+REPL_ID=
+REPL_SLUG=
+REPL_OWNER=
+REPLIT_DEV_DOMAIN=
+
+# Gemini (optional depending on features you want enabled)
+AI_INTEGRATIONS_GEMINI_API_KEY=
+AI_INTEGRATIONS_GEMINI_BASE_URL=https://generativelanguage.googleapis.com
+
+# Telegram (optional depending on features you want enabled)
+TELEGRAM_BOT_TOKEN=
+TELEGRAM_CHAT_ID=
+
+# Optional: tune AI robustness
+AI_TIMEOUT_MS=30000
+AI_MAX_RETRIES=2
```

README rewrite guidance: - Explicitly mention Replit's modules (`nodejs-20`, `postgresql-16`) and autoscale deployment context. [14]
- Mention which env vars are **hard required** in `server/index.ts` (`DATABASE_URL`, `SESSION_SECRET`) vs feature flags. [15]

Testing/verification steps - In Replit: `cp .env.example .env`, fill needed keys, `npm ci`, `npm run db:push`, `npm run dev` → server starts with no missing-required-var fatal. [16]
- Confirm README instructions match real scripts (`package.json`) and Drizzle push command. [17]

Estimated effort / suggested PR size - Effort: **low**
- PR size: **small** (~100–250 LOC, docs only)

Primary repo sources (raw URLs)

```
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/README.md
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/.env.example
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/index.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/.replit
```

PR 2: Fix HTML escaping and stripping (trust boundary)

Concise description
Fix `escapeHtml()` and `stripHtml()` in `server/utils/html.ts`. Ensure output is safe for Telegram's `parse_mode: "HTML"`.

Why it matters (impact/risk)
Telegram messages are sent with `parse_mode: "HTML"`. [18]
But `escapeHtml()` is currently broken (invalid replace patterns/quoting; does not escape `&`, `<`, `>` correctly). This is high risk: malformed messages, broken formatting, and potential HTML interpretation of user/model content. [3]

Exact files/paths to change - `server/utils/html.ts` [19]
- Verify call sites that depend on escaping: - `server/telegram.ts` [18]
- `server/telegram-bot.ts` [20]
- `server/routes/scan.ts` [21]
- `server/reddit-monitor.ts` / `server/google-alerts-monitor.ts` [22]

Small diff example

```
diff --git a/server/utils/html.ts b/server/utils/html.ts
--- a/server/utils/html.ts
+++ b/server/utils/html.ts
@@
 export function escapeHtml(text: string): string {
-  return text
-    .replace(/&/g, "&")
-    .replace(//g, ">")
-    .replace(/"/g, """)
-    .replace(/'/g, "'");
+  return text
+    .replace(/&/g, "&amp;")
+    .replace(/</g, "&lt;")
```

```
+    .replace(/>/g, "&gt;")
+    .replace(/"/g, "&quot;")
+    .replace(/'/g, "&#39;");
 }

 export function stripHtml(html: string): string {
-  return html
-    .replace(/<[^>]*>/g, " ")
-    .replace(/ /g, " ")
-    .replace(/&/g, "&")
-    .replace(/</g, "<")
-    .replace(/>/g, ">")
-    .replace(/"/g, '"')
-    .replace(/'/g, "'")
-    .replace(/\s+/g, " ")
-    .trim();
+  // Minimal: strip tags, normalize whitespace. (If you later need full entity
decoding, add a library.)
+  return html
+    .replace(/<[^>]*>/g, " ")
+    .replace(/\u00A0/g, " ")
+    .replace(/\s+/g, " ")
+    .trim();
 }
```

Optional (library-backed) improvement if you later want accurate entity decoding instead of partial custom logic: - Consider a dedicated HTML entity decoder (unspecified to avoid forcing deps). If you do add one, cite its official docs.

Testing/verification steps - Add unit tests to prove escaping correctness for `& < > " '` and ensure no Telegram HTML breakage. - Manual smoke: trigger `sendTelegramMessage("Test <b>bold?</b> & stuff")` and confirm the message displays literal `<b>` rather than bold formatting unless intended. This directly validates safety under `parse_mode: "HTML"`. [18]

Estimated effort / suggested PR size - Effort: **tiny/low** - PR size: **small** (~30–120 LOC plus tests)

Primary repo sources (raw URLs)

```
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/html.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram-bot.ts
```

PR 3: Make AI JSON parsing typed, robust, and consistently validated

Concise description
Refactor `parseAIJsonWithRetry` to be fully generic (`<T>`), require a `ZodSchema<T>`, and never return unvalidated data. Improve `safeParseJsonFromAI` to handle common model outputs. Update scan/Telegram flows to use schema validation.

Why it matters (impact/risk)
Your product depends on AI returning JSON reliably (lead scoring, strategy generation). You already have retry/timeouts and schemas, but the parsing method is a regex extract for `{...}` and the function signatures are underspecified. [23]
In addition, some flows bypass schema validation (e.g., scan endpoints use `safeParseJsonFromAI` directly). [21]
For a SaaS, schema-first parsing reduces production crashes and "silent weirdness."

Exact files/paths to change - `server/utils/ai.ts` [24]
- Update call sites where schema validation is missing or inconsistent: - `server/routes/scan.ts` (currently uses `safeParseJsonFromAI` directly) [21]
- `server/telegram/analysis.ts` (uses AI parsing for matching; likely similar pattern) [25]
- Keep monitors using `parseAIJsonWithRetry` + `leadScoreSchema` (already good) [22]

Small diff example

```
diff --git a/server/utils/ai.ts b/server/utils/ai.ts
--- a/server/utils/ai.ts
+++ b/server/utils/ai.ts
@@
-import { z, ZodSchema } from "zod";
+import { z, ZodSchema } from "zod";
@@
-export function safeParseJsonFromAI(text: string): any | null {
-  const cleaned = text.replace(/```json\s*/g, "").replace(/```\s*/g,
"").trim();
-  const jsonMatch = cleaned.match(/\{[\s\S]*\}/);
-  if (!jsonMatch) return null;
-  try { return JSON.parse(jsonMatch[0]); } catch { return null; }
-}
+export function safeParseJsonFromAI(text: string): unknown | null {
+  const cleaned = text
+    .replace(/```json\s*/gi, "")
+    .replace(/```\s*/g, "")
+    .trim();
+
+  // First try: whole-string JSON
+  try { return JSON.parse(cleaned); } catch {}
+
```

```
+  // Fallback: extract first JSON object or array block
+  const match = cleaned.match(/(\{[\s\S]*\}|\[[\s\S]*\])/);
+  if (!match) return null;
+  try { return JSON.parse(match[1]); } catch { return null; }
+}
@@
-export async function parseAIJsonWithRetry(
-  generateFn: () => Promise,
-  schema: ZodSchema,
-  maxRetries: number = 1
-): Promise {
+export async function parseAIJsonWithRetry<T>(
+  generateFn: () => Promise<string>,
+  schema: ZodSchema<T>,
+  maxRetries: number = 1
+): Promise<T | null> {
   for (let attempt = 0; attempt <= maxRetries; attempt++) {
     try {
       const text = await generateFn();
       const parsed = safeParseJsonFromAI(text);
       if (!parsed) {
@@
-      const validated = schema.safeParse(parsed);
+      const validated = schema.safeParse(parsed);
       if (validated.success) {
         return validated.data;
       }
@@
-      return null;
+      return null;
     } catch (err) {
@@
   return null;
 }
```

Then update scan flow to require schema validation:

```
diff --git a/server/routes/scan.ts b/server/routes/scan.ts
--- a/server/routes/scan.ts
+++ b/server/routes/scan.ts
@@
-import { generateContent, safeParseJsonFromAI, TONE_MAP, MIN_POST_LENGTH,
MIN_SCAN_INTENT_SCORE } from "../utils/ai";
+import { generateContent, parseAIJsonWithRetry, leadScoreSchema, TONE_MAP,
MIN_POST_LENGTH, MIN_SCAN_INTENT_SCORE } from "../utils/ai";
@@
```

```
-const match = safeParseJsonFromAI(matchResult.text);
+const match = await parseAIJsonWithRetry(
+  async () => matchResult.text,
+  leadScoreSchema,
+  1
+);
 if (!match) {
   ...
 }
```

Testing/verification steps - Unit tests (new) for: - `safeParseJsonFromAI` handles: raw JSON, ``` ```json ``` fences, leading/trailing prose.
- parseAIJsonWithRetry<T> `returns` T `only when Zod validates; otherwise returns` null`. - Integration smoke: - Run scan endpoint workflow and confirm it still returns “matched/low_intent” paths without crashing on model drift. [21]
- Run monitor pipeline (Reddit/Alerts) and confirm it still generates/sends messages. [22]

Estimated effort / suggested PR size - Effort: **medium** - PR size: **medium** (~200–500 LOC depending on number of call sites)

Primary repo sources (raw URLs)

```
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/ai.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/routes/scan.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/reddit-monitor.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/google-alerts-
monitor.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram/
analysis.ts
```

External official docs (when implementing changes)

```
https://zod.dev/
https://ai.google.dev/
```

[26]

PR 4: SaaS-grade package.json metadata + scripts

Concise description
Rename the project from “rest-express” to “gemin-eye”, add repository metadata, add `lint` and `test` scripts, and split `check` into `typecheck` alias for clarity.

Why it matters (impact/risk)
Branding/packaging matters when selling. Right now `package.json` still reads like a template (`"name": "rest-express"`) and has no lint/test scripts (only `check` + build/dev). [27]

Exact files/paths to change - `package.json` [28]
- (Potentially) `tsconfig.json` if you want test files typechecked (currently tests excluded). [29]

Small diff example

```
diff --git a/package.json b/package.json
--- a/package.json
+++ b/package.json
@@
-  "name": "rest-express",
+  "name": "gemin-eye",
   "version": "1.0.0",
   "type": "module",
   "license": "MIT",
+  "description": "AI-powered customer acquisition platform for monitoring
communities and generating lead responses",
+  "repository": {
+    "type": "git",
+    "url": "git+https://github.com/kmages/Gemin-Eye.git"
+  },
+  "bugs": { "url": "https://github.com/kmages/Gemin-Eye/issues" },
+  "homepage": "https://github.com/kmages/Gemin-Eye#readme",
+  "engines": { "node": ">=20" },
   "scripts": {
     "dev": "NODE_ENV=development tsx server/index.ts",
     "build": "tsx script/build.ts",
     "start": "NODE_ENV=production node dist/index.cjs",
-    "check": "tsc",
+    "check": "tsc",
+    "typecheck": "tsc",
+    "lint": "eslint .",
+    "test": "vitest run",
+    "test:watch": "vitest"
     "db:push": "drizzle-kit push"
   }
```

Testing/verification steps - `npm run typecheck` works (aliases `tsc`). [30]
- After PR 5 adds ESLint/Vitest configs, `npm run lint` and `npm test` pass.

Estimated effort / suggested PR size - Effort: **low** - PR size: **small** (~30–80 LOC)

Primary repo sources (raw URLs)

```
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/package.json
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/tsconfig.json
```

PR 5: Add CI + lint + tests (guardrails before refactors)

Concise description
Add GitHub Actions CI (Node 20, `npm ci`), ESLint (TypeScript-aware), and Vitest with a small starter suite focused on your highest-risk utilities: HTML escaping, AI JSON parsing, URL canonicalization, and rate-limiting behavior.

Why it matters (impact/risk)
This gives you a safety net before you do deeper refactors (service separation, rate limit persistence). The repo currently has no CI workflow files (none found under `.github/workflows/*`). [31]

Exact files/paths to change - `.github/workflows/ci.yml` (new)
- `eslint.config.mjs` (new; "flat config")
- `package.json` (scripts/devDeps) [28]
- `server/utils/*.test.ts` (new)
- Optional: keep `tsconfig.json` excluding tests; or introduce `tsconfig.test.json` if you want TSC typecheck of tests. [29]

Small code examples

GitHub Actions (Node 20, npm cache):

```
name: CI
on:
  pull_request:
  push:
    branches: [main]

jobs:
  build-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 20
          cache: npm
      - run: npm ci
      - run: npm run typecheck
```

```
      - run: npm run lint
      - run: npm test
```

ESLint config: start with typescript-eslint recommended presets (flat config). [32]

Vitest: Vite-native, works cleanly with TS/ESM projects. [33]

Testing/verification steps - In Replit shell: - `npm ci` - `npm run typecheck` - `npm run lint` - `npm test` - In GitHub: open PR → CI runs and blocks regressions.

Estimated effort / suggested PR size - Effort: **medium** (new tooling + first tests) - PR size: **medium** (~200–600 LOC)

External official docs

```
https://vitest.dev/
https://eslint.org/docs/latest/use/getting-started
https://typescript-eslint.io/getting-started/
```

[34]

PR 6: Persist rate limiting across autoscale and restarts

Concise description
Replace in-memory `createRateLimiter` with a store-backed limiter that can persist across autoscale instances (prefer Postgres since it's already required; allow opt-in Redis later).

Why it matters (impact/risk)
Your `.replit` deployment target is autoscale. [6]
Current limiter uses a process-local `Map` (`buckets`) and periodic cleanup. That will not enforce limits across instances, and resets on restarts—high risk for abuse/cost spikes on AI endpoints and Telegram webhooks. [35]

Exact files/paths to change - `server/utils/rate-limit.ts` [36]
- Call sites: - `server/routes.ts` (AI endpoints limiter) [37]
- `server/routes/scan.ts` (scan endpoints limiter) [21]
- `server/telegram-bot.ts` (webhook limiter) [20]
- DB schema addition (Postgres-backed limiter): - `shared/schema.ts` [38]
- plus migration creation (under `migrations/`, per `drizzle.config.ts`). [39]

Small diff example (concept level)

Add a new table (example fields): - `name` (limiter name) - `key` (client key) - `count` - `reset_at`

Then implement upsert logic in `createRateLimiter`.

If you later choose Redis store, node-redis is the official Redis Node client. [40]

Testing/verification steps - Unit tests: concurrent calls against same key should enforce 429 after threshold. - Manual: deploy to autoscale; hit `/api/strategy/generate` repeatedly and confirm rate limit holds across restarts.

Estimated effort / suggested PR size - Effort: **medium** - PR size: **medium** (~250–700 LOC including migration + tests)

Primary repo sources (raw URLs)

```
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/rate-
limit.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/shared/schema.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/.replit
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/routes.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram-bot.ts
```

External official docs (if adding Redis option)

```
https://github.com/redis/node-redis
```

[40]

PR 7: DB constraints and indexes tuned for monitoring + SaaS growth

Concise description
Your schema already has key foreign keys and several indexes; extend with a few high-value indexes for monitoring queries and growth (especially around campaign filtering and feedback lookup).

Why it matters (impact/risk)
`shared/schema.ts` already includes FKs and indexes on user/business/campaign IDs, as well as `seen_items.dedup_key` unique. [41]
However, monitors currently load **all businesses and all campaigns** and then filter in memory, which will degrade as you add more agencies/clients. [22]
Also, feedback joins in `getFeedbackGuidance` involve multiple joins; indexing `response_feedback.response_id` and possibly `(campaigns.business_id)` is useful (the latter already exists via `idx_campaigns_business_id`). [42]

Exact files/paths to change - `shared/schema.ts` (add indexes/constraints) [38]
- Migration files (generated by drizzle-kit) per `drizzle.config.ts`. [39]
- Optional monitor/query refactor: - `server/reddit-monitor.ts` and `server/google-alerts-monitor.ts` to filter at SQL level instead of in JS. [22]

Example additions - Index `campaigns.platform`, `campaigns.status` (or composite `(platform, status)`), since monitors filter by those fields. [43]
- Index `response_feedback.response_id` for feedback lookups and uniqueness checks. [44]
- Add `seen_items.created_at` index if you plan cleanup/retention jobs (optional). [38]

Testing/verification steps - `npm run db:push` applies cleanly on a new DB. [45]
- Sanity-check monitor queries still return expected targets.

Estimated effort / suggested PR size - Effort: **medium** - PR size: **medium** (~150–500 LOC including migrations)

Primary repo sources (raw URLs)

```
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/shared/schema.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/reddit-monitor.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/google-alerts-monitor.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/feedback.ts
```

PR 8: Route/service separation with shared lead-processing services

Concise description
Move business logic (AI prompts, matching logic, DB writes) out of route handlers and into reusable services so monitors, scan routes, Telegram analysis, and API endpoints share one authoritative pipeline.

Why it matters (impact/risk)
You currently have similar logic repeated in: - `server/routes.ts` (strategy generation, response generation, lead scoring) [46]
- `server/routes/scan.ts` (scan + match + response) [21]
- `server/telegram/analysis.ts` (match + response + DB save) [47]
- monitors (lead score + response + dedup) [22]

Service separation reduces drift and makes it practical to add tests and plan gating (rate limits, per-tenant quotas) later.

Exact files/paths to change - Create new directory: `server/services/` - `server/services/leadMatching.ts` (keyword filter + AI lead score) - `server/services/responseGeneration.ts` (platform-specific response prompts) - `server/services/leadPersistence.ts` (DB writes for lead +

ai_response) - Update call sites: - `server/routes.ts` [48]
- `server/routes/scan.ts` [21]
- `server/telegram/analysis.ts` [49]
- `server/reddit-monitor.ts`, `server/google-alerts-monitor.ts` [22]

Service/module boundary diagram

```
flowchart TB
  subgraph Routes
    RT[server/routes.ts]
    RS[server/routes/scan.ts]
    RA[server/routes/admin.ts]
  end

  subgraph Monitors
    RM[server/reddit-monitor.ts]
    GM[server/google-alerts-monitor.ts]
  end

  subgraph Telegram
    TA[server/telegram/analysis.ts]
  end

  subgraph Services
    LM[services/leadMatching]
    RG[services/responseGeneration]
    LP[services/leadPersistence]
  end

  subgraph Infra
    AI[utils/ai.ts]
    HTML[utils/html.ts]
    RL[utils/rate-limit.ts]
    DB[(server/db.ts + shared/schema.ts)]
  end

  RT --> LM
  RS --> LM
  RM --> LM
  GM --> LM
  TA --> LM

  LM --> AI
  RG --> AI
  LP --> DB
```

```
  RS --> RL
  RT --> RL
  TA --> RL

  RT --> HTML
  TA --> HTML
```

Testing/verification steps - Add at least 1 integration test around the extracted service boundary (e.g., lead matching returns “no match” when keywords don’t match). - Smoke test user paths: - Dashboard: `/api/leads`, response generation [50]
- Scan endpoints: `/api/fb-scan` or `/api/li-scan` (routes indicated in README) [51]
- Telegram bot happy path: `/api/telegram/webhook/<token>` still processes updates. [20]

Estimated effort / suggested PR size - Effort: **high** - Suggested PR size: **medium-to-large**, but split by domain (first extract lead matching; then response gen; then persistence).

Primary repo sources (raw URLs)

```
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/routes.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/routes/scan.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram/
analysis.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/reddit-monitor.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/google-alerts-
monitor.ts
```

Proposed PR sequence with dependencies, checklists, graph, and PR comparison table

PR dependency graph

```
graph TD
  P1["PR1: README + .env.example formatting"] --> P5
  P2["PR2: Fix escapeHtml/stripHtml"] --> P5
  P3["PR3: Typed AI JSON parsing + consistent validation"] --> P5
  P4["PR4: package.json metadata + scripts"] --> P5

  P5["PR5: CI + ESLint + Vitest + starter tests"] --> P6
  P5 --> P7
  P5 --> P8

  P6["PR6: Persistent rate limiting"] --> P8
```

```
  P7["PR7: DB index/constraint tuning"] --> P8
  P8["PR8: Route/service separation"]
```

Checklist per PR (agentic Replit coder oriented)

PR 1 checklist - [ ] Reformat `README.md` into readable Markdown (real headings, lists, code fences). - [ ] Reformat `.env.example` into multiline `.env` format (one KEY=value per line). - [ ] Confirm README matches Node 20 requirement (Replit config uses Node 20). [14]
- [ ] Replit verification: run `npm ci`, `cp .env.example .env`, `npm run db:push`, `npm run dev`. [52]

PR 2 checklist - [ ] Fix `escapeHtml` and `stripHtml` in `server/utils/html.ts`. [19]
- [ ] Manual test: send Telegram message containing `<` and `&` and verify no formatting injection under `parse_mode: "HTML"`. [18]

PR 3 checklist - [ ] Implement `parseAIJsonWithRetry<T>` typed generic and robust JSON extraction. [24]
- [ ] Update `server/routes/scan.ts` to use schema validation for lead score results. [53]
- [ ] Update Telegram analysis parsing path similarly if it uses raw parsing. [25]

PR 4 checklist - [ ] Rename package name from `rest-express` → `gemin-eye`. [28]
- [ ] Add `typecheck`, `lint`, `test` scripts. - [ ] Add `engines.node` = `>=20` to match Replit config. [6]

PR 5 checklist - [ ] Add GitHub Actions workflow (Node 20, `npm ci`, typecheck/lint/test). - [ ] Add ESLint + typescript-eslint flat config. [32]
- [ ] Add Vitest and 4–6 targeted unit tests (HTML escaping, AI parse, canonicalizeUrl, rate-limit). [54]

PR 6 checklist - [ ] Add persistent limiter backend (default Postgres). - [ ] Update `createRateLimiter` call sites: routes, scan, telegram webhook. [55]
- [ ] Add tests validating persistence semantics (at least functional behavior).

PR 7 checklist - [ ] Add indexes/constraints where missing (platform/status indexes, feedback index). - [ ] Optionally refactor monitors to query only the needed campaigns from DB (avoid in-memory filtering). [22]

PR 8 checklist - [ ] Add `server/services/*` and migrate one pipeline end-to-end (recommend starting with scan pipeline because it's self-contained). [56]
- [ ] Ensure monitors and Telegram analysis call the same services (no duplicated prompts). - [ ] Add 1–2 integration tests around service boundary.

PR comparison table

PR #	Title	Files changed (representative)	Effort	Risk	Est. LOC
1	README + `.env.example` formatting	`README.md`, `.env.example`	low	low	100–250
2	Fix HTML escaping/stripping	`server/utils/html.ts` (+ tests later)	tiny/low	medium	40–150
3	Typed AI JSON parsing + consistent validation	`server/utils/ai.ts`, `server/routes/scan.ts`, `server/telegram/analysis.ts`	medium	medium	200–500
4	package.json metadata + scripts	`package.json`	low	low	30–80
5	CI + ESLint + Vitest + starter tests	`.github/workflows/ci.yml`, `eslint.config.mjs`, tests	medium	low/med	250–700
6	Persistent rate limiting	`shared/schema.ts`, migrations, `server/utils/rate-limit.ts`, call sites	medium	medium/high	250–700
7	DB constraint/index tuning + monitor query cleanup	`shared/schema.ts`, monitor files	medium	medium	150–600
8	Route/service separation	`server/services/*`, route/monitor/telegram call sites	high	high	500–2000

Testing and verification plan for Replit and CI

Replit verification commands (agent-friendly)

After each PR:

```
npm ci
npm run typecheck
npm run lint
npm test
npm run db:push
npm run dev
```

These map to your current scripts (`dev`, `db:push`, `check`) and the proposed additions (`typecheck`, `lint`, `test`). [52]

What to test first (risk-based)

- **Trust boundary**: Telegram message formatting under `parse_mode: "HTML"` with hostile strings (contains `<`, `&`, quotes). [57]
- **AI contracts**: lead scoring JSON schema and retry handling across monitors and scan endpoints. [58]
- **Autoscale behavior**: persistent rate limiter in production mode (multiple instances) vs in-memory limiter. [59]

CI defaults and notes

- CI provider assumed: **GitHub Actions** (requested default).
- Node version: repo is explicitly Node 20 in Replit and README; CI should align to Node 20. [14]
- ESLint prerequisites: ESLint's docs list modern Node requirements; keep CI on Node 20+ to match. [60]

Appendix of primary repo sources

Raw URLs requested (in code blocks):

```
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/.replit
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/README.md
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/.env.example
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/package.json
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/tsconfig.json
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/drizzle.config.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/index.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/routes.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/routes/admin.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/routes/scan.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/html.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/ai.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/rate-
limit.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/dedup.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/feedback.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/shared/schema.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram-bot.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/reddit-monitor.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/google-alerts-
monitor.ts
```

External official docs referenced:

```
https://zod.dev/
https://vitest.dev/
https://eslint.org/docs/latest/use/getting-started
https://typescript-eslint.io/getting-started/
https://github.com/redis/node-redis
```

61

---

1 https://github.com/kmages/Gemin-Eye
https://github.com/kmages/Gemin-Eye

2 9 11 51 raw.githubusercontent.com
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/README.md

3 19 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/html.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/html.ts

4 23 24 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/ai.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/ai.ts

5 6 14 59 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/.replit
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/.replit

7 15 16 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/index.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/index.ts

8 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/db.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/db.ts

10 13 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/replit.md
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/replit.md

12 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/.env.example
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/.env.example

17 27 28 30 52 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/package.json
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/package.json

18 57 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram.ts

20 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram-bot.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram-bot.ts

21 53 56 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/routes/scan.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/routes/scan.ts

22 58 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/reddit-monitor.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/reddit-monitor.ts

25 47 49 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram/analysis.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram/analysis.ts

26 61 https://zod.dev/
https://zod.dev/

29 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/tsconfig.json
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/tsconfig.json

31 (no title)

32 https://typescript-eslint.io/getting-started/
https://typescript-eslint.io/getting-started/

33 34 54 https://vitest.dev/
https://vitest.dev/

35 36 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/rate-limit.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/rate-limit.ts

37 46 48 50 55 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/routes.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/routes.ts

38 41 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/shared/schema.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/shared/schema.ts

39 45 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/drizzle.config.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/drizzle.config.ts

40 https://github.com/redis/node-redis
https://github.com/redis/node-redis

42 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/feedback.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/utils/feedback.ts

43 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/google-alerts-monitor.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/google-alerts-monitor.ts

44 https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram/callbacks.ts
https://raw.githubusercontent.com/kmages/Gemin-Eye/main/server/telegram/callbacks.ts

60 https://eslint.org/docs/latest/use/getting-started
https://eslint.org/docs/latest/use/getting-started