

Submit Your SaaS Project

Grade: A

Status: Pass (automatic by LLM)

Feedback

Lubo -- fantastic work. MergeAI feels like a real product with a crisp value prop, a believable multi-agent architecture, and thoughtful transparency around what the system is doing. Your technical choices (App Router route handlers, SSE streaming, Drizzle + Neon, NVIDIA models via OpenAI SDK) are solid and well-structured.

Evaluation checklist

- Landing page screenshot: received and clear
- Routes file: I see multiple route.ts files that appear to correspond to /api/files (GET), /api/query (POST, SSE), and /api/upload (POST). Please confirm the exact file paths under app/api so I can record them precisely.
- Live deployment URL: provided. I wasn't able to fully exercise the flow in this environment; if a test account/demo user or "try without login" path exists, please share instructions.
- GitHub repo: provided and appears complete
- Submission form: received and clear

Category scores and feedback

1) Landing Page Quality -- 24/25

- What worked: Clear headline and subhead ("Upload. Ask. Watch AI Think."), crisp value prop, strong CTA, and a believable "How it works" section that mirrors your agent pipeline. Visual hierarchy feels professional.
- Suggestions: Add a concise, skimmable proof point right under the hero (e.g., "Joins across files automatically. SQL you can inspect.") and a tiny secondary CTA like "Try with sample data" for immediate engagement.

2) Next.js Route Architecture -- 19/20

- What worked: Clean separation of concerns and a coherent product story from the routes alone:
- Likely /api/upload (POST) to ingest rows
- Likely /api/files (GET) to list user and demo files
- Likely /api/query (POST with SSE) to orchestrate agents and stream progress
- Clerk middleware protects /dashboard and /settings
- Suggestions:
 - Security: API routes currently trust a userId passed from the client. Derive userId server-side via Clerk (auth()) in the route handlers or protect /api/* via middleware, then remove userId from request bodies/queries. This avoids cross-user data access by spoofed IDs.

3) Core Functionality -- 22/25

- What worked: The end-to-end path is thoughtfully designed: upload CSV -> schema agent proposes a join -> SQL agent generates real Postgres over JSONB -> validator enforces basic data sanity -> SSE updates the UI and returns a readable summary. The retry loop with actionable feedback is excellent for resilience in a 48-hour build.
- Suggestions:
 - I couldn't fully verify the flow live here. Please share:

- A test account or a "demo mode" link that preloads demo files and bypasses auth
- 1-2 sample CSVs and a quick set of questions to reproduce a successful join
- Consider limiting row counts or adding statement timeouts to ensure queries return quickly and don't stall the SSE.

4) Technical Implementation -- 13/15

- What worked:
- Agents are cleanly separated (schema/sql/validator/summary) with a clear orchestrator and event bus.
- Streaming via ReadableStream + SSE is implemented cleanly in /api/query and parsed nicely in the useAgentStream hook.
- Drizzle schemas are tidy; Neon integration is straightforward; types are clear.
- Risks and improvements:
- Auth trust boundary: Don't accept userId from the client. Use Clerk server-side auth() and enforce in the handlers. Optionally allow a demo path that sets isDemo=true and omits user context altogether.
- LLM-generated SQL safety: You execute unverified dynamic SQL strings.

Mitigations to add quickly:

- Create a DB role that is strictly read?only (no DDL/DML), set default_statement_timeout, and lock search_path.
- Enforce a strict allowlist: only SELECT; reject queries containing semicolons, INSERT/UPDATE/DELETE/ALTER/DROP/TRUNCATE, COPY, DO, CALL, WITH DATA/NO DATA into, etc.
- Enforce an upper bound LIMIT (e.g., 200) and reject queries without it.
- Scalability: You post parsed rows as JSON to the server. For larger files, switch to server?side parsing or streaming upload to object storage and then ingest (COPY, or batched inserts) to avoid memory spikes.

5) Problem & Idea Clarity -- 10/10

- Exceptionally clear articulation: who it's for, what's different, and how the agents collaborate. The concrete example ("EmpID" vs "Employee ID", CTEs, JSONB, case insensitive joins) lands well.

6) Polish & Completeness -- 4/5

- The product feels cohesive. The agent status stream + copyable SQL is a strong trust builder.
- Minor gaps: ensure favicon/loading/error states across the app; add a one-click demo; confirm responsive behavior across breakpoints.

Notable strengths

- Transparent agent pipeline with retries and diagnosable messages.
- Real SQL over a JSONB data lake pattern; pragmatic for a hackathon.
- Clean UI and clear storytelling on the landing page.

High-impact next steps

- Security and auth:
- Derive userId from Clerk in route handlers; optionally protect /api/* via middleware.
- Add read-only DB role + SQL allowlist and statement timeouts.
- Performance/scale:
- Move to server-side CSV parsing or streaming uploads; consider materializing normalized join keys (e.g., lower()ed IDs) and adding expression indexes for common joins.

- UX:

- Add a "Try with sample data" flow and a short Loom showing the end-to-end run.
- Let users manually override/confirm the inferred join when validation retries fail.

What I need from you to finalize/verify anything I couldn't test here

- A test user (or demo-mode link) and quick steps to reproduce a successful query.
- Confirmation of the exact route paths (e.g., app/api/files/route.ts, app/api/upload/route.ts, app/api/query/route.ts).
- If applicable, sample CSVs you used and 2-3 example questions.

Preliminary score summary

- Landing Page Quality: 24/25
- Route Architecture: 19/20
- Core Functionality: 22/25
- Technical Implementation: 13/15
- Problem & Idea Clarity: 10/10
- Polish & Completeness: 4/5

Total: 92/100

The following files were not recognized (unknown format): page.tsx (.tsx), page.tsx (.tsx), favicon.ico (.ico), page.tsx (.tsx), layout.tsx (.tsx), page.tsx (.tsx), page.tsx (.tsx).

For grading, supported formats are: documents as PDF or plain text (.txt, .md, .rtf); images as .png, .jpg, .jpeg, .gif, .webp. Submit as a .zip or a single image.