Production-Grade AI Data Cleaning API Roadmap

Overview

This project builds a high-performance, asynchronous REST API for conversational data transformation. The system uses a **3-Agent Crew** (Strategist, Engineer, Tester) to intelligently modify datasets based on user chat commands.

Unlike simple MVPs, this architecture is designed for **Scale and Reliability**. It features a dedicated **Data Profiling Engine** to give the AI vision, a **Secure Code Sandbox** to prevent malicious execution, and a fully decoupled **Async Worker Queue (Celery/Redis)** to handle heavy traffic without blocking.

Tools & Technologies

- **Backend:** FastAPI (Python 3.10+).
- **Orchestration:** CrewAI (Multi-Agent System).
- **Async Queue:** Celery & Redis.
- **Database:** Supabase (PostgreSQL).
- **File Storage:** Supabase Storage (S3-compatible).
- **Data Engine:** Pandas & NumPy.
- **Deployment:** Railway (Dockerized Multi-Service).

Project Milestones

1. System Architecture & Database Schema

Before writing code, we must define the data models that support "Time Travel" (Undo/Redo) and complex conversation history.

- **Process:**
    1. **Schema Design:** Design the SQL schema in Supabase.
        - `sessions`: The container for the interaction.
        - `nodes`: A Linked List structure (parent_id -> child_id) to track every version of the CSV.
        - `chat_logs`: Linking specific messages to specific Nodes.
    2. **API Contract:** Define the JSON Request/Response models for every endpoint (using Pydantic).
    3. **Repository Setup:** Initialize the Git repo with a modular structure: `/app`, `/core`, `/agents`, `/workers`.
- **Output:** Validated Database Schema and OpenAPI Specification.
- **Time:** 8 hours
- **Cost:** $320

2. The Data Profiling Intelligence

The AI cannot "see" the CSV. We must build a Python engine that translates raw data into a language the LLM understands.

- **Process:**
    1. **Ingestion Utility:** Build a robust CSV reader that handles edge cases (bad delimiters, encoding errors, huge files).
    2. **Profiling Logic:** Create `generate_profile(df)`. It must extract:
        - Column Names & Types.
        - Sample Values (First 3 rows).
        - Null Value Analysis ("Column 'Age' has 20% missing").
        - Outlier Detection (Optional but high value).
    3. **Token Optimization:** Ensure this summary is concise enough to fit in the AI's context window.
- **Output:** A standalone Python module that turns any CSV into a "Text Summary" for the Agent.
- **Time:** 10 hours
- **Cost:** $400

3. Agent A: The "Context Strategist" (Logic Revamp)

Rebuilding the "Brain" to handle conversation history and intent.

- **Process:**
    1. **Memory Integration:** Build the logic to fetch the last N messages from Supabase and format them for the Agent.
    2. **System Prompt Engineering:** Write the specific instructions for the Strategist.
        - *Goal:* "Map user intent ('drop bad rows') to specific columns based on the Data Profile."
    3. **Referential Understanding:** Test the agent's ability to understand "Undo that" or "Do the same for column X."
- **Output:** The `Strategist` agent capable of outputting a clear Technical Plan from vague chat.
- **Time:** 12 hours
- **Cost:** $480

4. Agent B & C: Code Generation & Security Sandbox

The "Hands" of the system. We must ensure the code is correct and SAFE to run.

- **Process:**
    1. **The Engineer (Agent B):** prompt it to write *only* the Pandas snippet, not a full script.
    2. **The Tester (Agent C):** A secondary LLM call to review the code for syntax errors.
    3. **Execution Sandbox (Crucial):**
        - Build a custom `exec()` wrapper.

    - **Block Dangerous Imports:** Explicitly forbid `os`, `sys`, `subprocess`, or `network` calls to prevent the AI from hacking the server.
    - **Scope Management:** Ensure the code only sees the dataframe variable `df`.
- **Output:** A secure function `execute_safe_transformation(code, df)` that returns a new dataframe or an error.
- **Time:** 14 hours
- **Cost:** $560

5. API Layer 1: Sessions & File Management

Building the synchronous "Front Door" of the API.

- **Process:**
    1. **FastAPI Setup:** Initialize the application with CORS and Error Handlers.
    2. **Session Endpoints:** `POST /session/create`.
    3. **Upload Pipeline:** `POST /upload`.
        - Stream file to Supabase Storage.
        - Run the **Data Profiling Engine** (Milestone 2).
        - Create "Node 0" in the DB.
- **Output:** A working API where you can upload a file and get a Session ID.
- **Time:** 8 hours
- **Cost:** $320

6. Async Infrastructure: Redis & Celery

Setting up the heavy-lifting plumbing.

- **Process:**
    1. **Redis Setup:** Configure the message broker on Railway.
    2. **Celery Configuration:** Set up the Task Queue logic.
    3. **Worker Dockerfile:** Create a specific Docker image that runs the Celery Worker (distinct from the Web API image).
    4. **Serialization:** Ensure that complex objects (like DataFrames or Supabase responses) can be passed through Redis without errors.
- **Output:** A working "Ping-Pong" test where the API sends a background task and the Worker picks it up.
- **Time:** 10 hours
- **Cost:** $400

7. API Layer 2: The Async Chat Loop

Connecting the API (M5) to the Agents (M3/M4) via the Queue (M6).

- **Process:**
    1. **`POST /chat`:** The endpoint that triggers the flow. It pushes the `session_id` and `message` to Redis.
    2. **The Worker Task:**
        - Fetch latest CSV from Storage.
        - Fetch History from DB.
        - Run Agents (Strategist -> Engineer -> Sandbox).
        - Save Result (New CSV) to Storage.
        - Update DB with new Node.
    3. **Polling Mechanism:** `GET /task/{id}` to report progress ("Thinking...", "Generating Code...", "Done").
- **Output:** The complete End-to-End flow. Chat in -> Wait -> Result out.
- **Time:** 18 hours
- **Cost:** $720

8. State Management: History & Rollbacks

Giving the user control over the timeline.

- **Process:**
    1. **History Endpoint:** `GET /history/{session_id}`. Returns the full conversation and the list of file versions.
    2. **Undo Logic:** `POST /node/rollback`.
        - Updates the "Current Pointer" of the session to a previous Node ID.
        - Ensures the next chat message builds off the *old* data, effectively branching the timeline.
- **Output:** Full version control capabilities for the dataset.
- **Time:** 8 hours
- **Cost:** $320

9. Production Hardening, Testing & Docs

Ensuring it doesn't break when the client uses it.

- **Process:**
    1. **Load Testing:** Simulate 20 concurrent users to tune the Celery Worker count.
    2. **Postman Collection:** Write a detailed "How-To" JSON collection for the client.
    3. **Deployment:** Final push to Railway (Web Service + Worker Service + Redis).

4. **Swagger Docs:** Add descriptions and example bodies to the auto-generated docs.

- **Output:** Live URL and Handoff Materials.
- **Time:** 12 hours
- **Cost:** $480

Time & Cost Summary

- **Total Time:** 95 Hours
- **Total Cost:** $3,800

Milestone	Description	Time (h)	Cost ($)
1. Architecture	DB Schema, API Contract, Repo Setup	8	$320
2. Profiling Engine	CSV Ingestion & Metadata Analysis logic	10	$400
3. Agent A (Strategist)	Memory Integration, Intent Prompts	12	$480
4. Agent B/C (Sandbox)	Code Gen, Safety Exec Wrapper	14	$560
5. API Foundation	Uploads, Auth, Storage Integration	8	$320
6. Async Infra	Redis, Celery, Docker Config	10	$400
7. Async Chat Loop	Connecting API to Workers	18	$720
8. State Manager	Undo/Redo, History Retrieval	8	$320
9. Deployment	Testing, Postman, Railway Push	12	$480
TOTAL		**100**	**$4000**

Final Deliverables

1. **The API Codebase:** A Python repository with clean separation of concerns (`api/`, `worker/`, `lib/`).
2. **The "Brain" Modules:** The custom logic for Data Profiling and Safe Code Execution.
3. **Live Production Environment:** Railway deployment with Supabase & Redis connected.
4. **Postman "Control Panel":** A file allowing the client to test every feature immediately.
5. **Technical Documentation:** A guide explaining the 3-Agent architecture and how to scale workers.