Aterges: Backend Development Plan

Version: 1.0
Lead: Solopreneur (acting as Backend Developer / Architect)
Primary Goal: To build a secure, scalable, and intelligent API backend that serves as the core engine for the Aterges platform, orchestrating AI models and data agents.

1. Core Objective

To develop a robust API that handles user authentication, manages the core business logic, and orchestrates the complex interactions between the user's prompt, Large Language Models (LLMs), and various external data source APIs (the "Agents").

2. Guiding Principles

- **Security First:** The architecture must prioritize the secure handling of user data and, most critically, the credentials provided by customers (service account keys).
- **Scalability by Design:** We will use a serverless and container-based approach to ensure the infrastructure can handle growth without manual intervention.
- **Modularity (The Hub & Spoke Model):** The core logic (the "Orchestrator") will be decoupled from the data connectors (the "Agents"). This allows for the independent development and addition of new data sources without affecting the core application.
- **Cost-Efficiency:** All architectural choices must favor a low-cost, pay-as-you-go model to maximize capital efficiency in the early stages.

3. Core Technical Stack

This stack is final and should be implemented with the specified versions to ensure security and stability.

- **Language: Python** (v3.13.3+)
- **Framework: FastAPI** (v0.115.11+)
- **Database & Auth: Supabase** (PostgreSQL v17.5+, subject to Supabase's provided version)
- **Containerization: Docker Engine** (v28.1.1+)
- **Deployment: Google Cloud Run**
- **CI/CD: GitHub Actions**
- **AI Model (Initial): Google Gemini** (via Vertex AI)
- **Secrets Management: Google Secret Manager**

4. Development Roadmap

Phase 0: The API Skeleton & Authentication (Current Focus)

This phase runs in parallel with the frontend skeleton development. Its goal is to

provide a working, deployable API with a complete authentication system.

- **[ ] Repository Setup:** GitHub monorepo with a /backend directory.
- **[ ] Project Initialization:** Setup Python environment (venv), requirements.txt, and FastAPI app structure.
- **[ ] Supabase Integration:** Initialize the Supabase Python client to connect to the database and auth services.
- **[ ] Implement Authentication Endpoints:**
  - POST /auth/signup: Create a new user in Supabase.
  - POST /auth/login: Authenticate a user and return a JWT.
  - POST /auth/logout: Invalidate the user's session.
- **[ ] Implement Protected Endpoint:**
  - GET /api/me: A test endpoint that requires a valid JWT to return the user's info. This is critical for the frontend team to test their authentication flow.
- **[ ] Dockerize the Application:** Create a Dockerfile to package the FastAPI app for production.
- **[ ] CI/CD Pipeline:** Configure a GitHub Actions workflow to automatically build the Docker image and deploy it to Google Cloud Run on pushes to main.
- **[HITO]** The API is live and the frontend team can successfully authenticate users against it.

Phase 1: The AI Orchestrator & First Agent (Next Steps)

Once the user can log in, we make the application "think".

- **[ ] Integrate AI Model:** Connect the backend to the **Google Gemini API** via Vertex AI. Create a service module to handle all interactions with the LLM.
- **[ ] Build the First Data Agent (GoogleAnalyticsAgent):**
  - Create a Python class dedicated to interacting with the Google Analytics Data API.
  - Implement a flexible, **autonomous method** like query_ga4(metrics, dimensions, date_ranges) that can handle a variety of requests.
- **[ ] Implement the Core Orchestrator Logic:**
  - Develop the logic for the POST /api/query endpoint.
  - Implement the **"Tool Calling"** flow:
    1. Describe the available tools (e.g., query_ga4) to the Gemini model.
    2. Send the user's prompt to Gemini.
    3. Receive the structured "tool call" request from Gemini.
    4. Execute the corresponding agent's method with the parameters provided by the AI.
    5. Send the tool's output back to Gemini.

6. Receive the final, human-readable response and send it back to the frontend.

Phase 2: Core Application Features (In Parallel)

These endpoints are required to support the Phase 2 features of the frontend. They are simpler CRUD operations and can be developed alongside the more complex AI logic.

- **[ ] User Settings API:**
  - PUT /api/user/profile: Update user's name.
  - POST /api/user/change-password: Handle password changes.
- **[ ] BYOK Integrations API:**
  - **Crucial:** Implement the secure flow using **Google Secret Manager** for storing customer credentials.
  - POST /api/integrations: Endpoint to receive a customer's service-account.json, store it in Secret Manager, and save the reference in the database.
  - GET /api/integrations: List a customer's configured integrations.
  - DELETE /api/integrations/{id}: Remove an integration and delete the corresponding secret.
- **[ ] Conversation History API:**
  - Design the database schema for storing chats and messages.
  - Implement a full set of CRUD endpoints (GET, POST, DELETE) for managing conversations.

5. Key Deliverables

- A secure, scalable, and production-ready FastAPI application deployed on Google Cloud Run.
- A fully functional authentication system integrated with Supabase.
- A working AI Orchestrator capable of using "Tool Calling" with a Google Analytics agent to answer user queries.
- A complete set of API endpoints to support all Phase 2 frontend features, including secure credential management for the BYOK model.