Project Plan: Subscotia Vault Web Application Document Version: 1.0 Date: 11 July 2025

1. Project Goal & Core Philosophy Goal: To create a fast, efficient, and visually appealing web application to search, filter, and manage the Subscotia Sound plugin vault. The application will run locally on the studio's dedicated Linux server and be accessed from the main workstation.

Core Philosophy:

Local-First: The application and database will be self-hosted on the local network. No reliance on external web services for core functionality.

Performance: The interface must be fast and responsive, allowing for instant filtering and searching of the entire plugin library.

Professional UI/UX: The application should look and feel like a bespoke piece of professional studio software, not a generic web page.

Modular & Maintainable: The code will be clean, well-documented, and structured logically to allow for future expansion and easy maintenance.

2. Technology Stack All selected technologies are free, open-source, and well-suited for this project.

Backend: Python 3

Web Framework: Flask (A lightweight and powerful Python framework ideal for creating the local server and API)

Frontend: HTML5, CSS3, JavaScript (ES6+)

CSS Framework: Tailwind CSS (For rapid development of a modern, professional, and responsive user interface)

Icons: Lucide Icons (For a clean, consistent, and lightweight icon set)

3. Development Phases This project will be developed in distinct, sequential phases.

Phase 1: Backend Foundation & API (Goal: Get a basic server running that can serve the vault data)

Task 1.1: Set up the basic Flask server application structure on the Linux machine.

Task 1.2: Create a single API "endpoint" that reads the master vault.json file and sends its contents to the browser.

Task 1.3: Configure the server to be accessible from other machines on the local network.

Task 1.4: Create a basic index.html page that successfully loads and displays the raw JSON data from the server.

Phase 2: Frontend UI/UX - The "Tasty" Interface (Goal: Build a visually appealing, modern interface to display the data)

Task 2.1: Integrate Tailwind CSS and the "Inter" font into the project.

Task 2.2: Design and build the main application layout, including a header, a filter sidebar, and a main content area.

Task 2.3: Implement a "card-based" layout to display each plugin as a distinct visual element, replacing the basic HTML table.

Task 2.4: Implement a user-toggleable dark/light theme that is visually distinct and professional.

Task 2.5: Ensure the entire layout is fully responsive and usable on different screen sizes.

Phase 3: Advanced Functionality & Interactivity (Goal: Implement the powerful filtering and searching features)

Task 3.1: Implement a live-search text input that filters cards in real-time.

Task 3.2: Implement multi-select checkbox filters for families and tags. This will be the core of the powerful search functionality.

Task 3.3: Implement a "Detail View" modal that appears when a user clicks on a plugin card, showing all associated data for that entry.

Task 3.4: (Optional) Implement dynamic grouping controls to allow users to group results by developer or product.

Phase 4: Data Management Interface (Goal: Allow for adding and editing plugins directly from the web interface)

Task 4.1: Design and build an "Add New Plugin" form within the web interface.

Task 4.2: Create a new backend API endpoint that can receive data from this form.

Task 4.3: Implement the Python logic to safely add the new entry to the master vault.json file and create a backup.

Task 4.4: Ensure the main interface automatically refreshes to show the newly added plugin without requiring a manual page reload.

4. Review & Refinement This project plan will be reviewed at the beginning and end of each major phase. The modular nature of the plan allows for flexibility, and features can be tweaked, added, or reprioritized as we progress and discover new requirements.