Product Requirements Prompt (PRP)

Feature: Backend API for New Plugins

Version: 1.0

Objective

To modify the Flask backend (app.py) to include a new API endpoint capable of receiving new plugin data from the frontend form. Additionally, the frontend (index.html) will be updated to gather the form data and send it to this new endpoint upon submission.

Execution Plan

Phase 1: Backend Modification (app.py)

1. **Create New Route:**
    - A new function, add_plugin(), will be created and decorated with @app.route('/api/plugin', methods=['POST']). The methods=['POST'] argument is critical to ensure the endpoint only accepts data submission requests.
2. **Implement Data Handling:**
    - Inside the add_plugin() function:
        - Use Flask's request object to get the JSON data from the incoming request (data = request.get_json()).
        - Implement a try...except block to handle potential errors, such as malformed JSON.
        - **Verification Step:** Print the received data dictionary to the Flask console using a formatted print statement (e.g., print(f"--- New Plugin Data Received ---\n{data}\n--------------------------------")).
        - Return a JSON response to the client indicating success (e.g., return jsonify({'status': 'success'}), 200).

Phase 2: Frontend Modification (index.html)

The main script block will be updated to handle the form submission

logic.

1. **Update initAddPluginModal() Function:**
    - The existing submit event listener for the #add-plugin-form will be replaced with a new, more detailed implementation.
2. **Implement Form Submission Logic:**
    - The new event listener will:
        - **a. Prevent Default:** Call e.preventDefault() to stop the browser's default page reload on form submission.
        - **b. Gather Data:** Get the values from each input field in the form (#name-input, #developer-input, etc.).
        - **c. Process Data:**
            - Convert the comma-separated strings from the families and tags inputs into clean arrays of strings. This will involve splitting the string by commas and trimming any whitespace from each item.
            - Convert the value from the active checkbox into a proper boolean (true/false).
        - **d. Construct Payload:** Assemble all the processed data into a single JavaScript object that matches the vault_master.json schema.
        - **e. Send Request:** Use the fetch() API to send the data to the new /api/plugin endpoint. The fetch call will be configured with:
            - method: 'POST'
            - headers: { 'Content-Type': 'application/json' }
            - body: JSON.stringify(payload)
        - **f. Handle Response:** Check the response from the server. For now, a simple alert() or console log confirming success is sufficient.
        - **g. Close Modal:** Close the "Add New Plugin" modal upon

successful submission.

Final Review

- Confirm the new /api/plugin route is correctly defined in app.py and only accepts POST requests.
- Test submitting the form from the frontend.
- Verify that the complete, correctly formatted JSON data is printed to the Flask console in PyCharm.
- Ensure the frontend receives a success response and closes the modal after submission.

Approval Request: The Product Director is requested to review this PRP. Upon approval, the Code Engine will proceed with the execution phase.