# GH Issues Agent

## Agent Capabilities

1. ***Summarization***
   * Summarize issues by product, tag, or time period.
   * Highlight trends (e.g., recurring themes, top-requested enhancements).
2. ***Analytics***
   * Count of issues opened/closed in a timeframe.
   * Time-to-close metrics.
   * PM responsiveness (e.g., time to first comment).
3. ***Deduplication & Similarity Detection***
   * Flag similar or duplicate issues.
   * Suggest merging or linking related issues.
4. ***Tagging Intelligence***
   * Suggest missing or more accurate tags based on issue content.
5. ***CSM Interface***
   * When a CSM creates a new issue:
     + Suggest similar existing issues.
     + Indicate if the issue has been resolved or is in progress.
     + Recommend tags or next steps (e.g., escalate, wait for PM input).

## Implementation

#### 1. **Data Pipeline**

You already have a script that exports GitHub issues to CSV. That’s a great start. You could:

* Automate this export on a schedule (e.g., daily).
* Store the data in a lightweight database or just keep it in CSV for now.

#### 2. **LLM-Powered Backend**

Use an LLM (like GPT-4 or Claude) to:

* Ingest the CSV.
* Run classification (e.g., question vs. enhancement vs. support).
* Perform semantic similarity checks (for deduplication).
* Generate summaries and tag suggestions.

This can be done via:

* A Python backend using LangChain or LlamaIndex.
* Or a no-code/low-code tool like **Flowise**, **Zapier with OpenAI**, or **Retool**.

#### 3. **CSM-Facing Interface**

Options:

* **GitHub App or Bot**: When a CSM opens an issue, the bot comments with:
  + Similar issues.
  + Suggested tags.
  + Status of related issues.
* **Web App**: A simple form where CSMs paste their issue, and the app:
  + Checks for duplicates.
  + Suggests tags.
  + Submits to GitHub via API.

### Suggested Architecture

| **Component** | **Purpose** |
| --- | --- |
| **GitHub CSV Export** | Source of truth for issues, comments, tags |
| **Embedding Model (e.g., OpenAI, HuggingFace)** | Converts issue text into vectors |
| **Vector DB (e.g., Pinecone, Weaviate, FAISS, Chroma)** | Stores and retrieves embeddings |
| **LLM (e.g., GPT-4)** | Summarization, classification, tag suggestions |
| **Interface (GitHub App or Web UI)** | CSM interaction layer |

### ✅ 1. **Understand MCP Tool Basics**

An MCP tool is essentially a service that exposes functionality (like fetching GitHub issues) to an MCP client via a standardized protocol. It usually consists of:

* **Manifest file** (mcp.json) describing the tool (name, description, endpoints).
* **Server implementation** (your Python logic wrapped in MCP endpoints).
* **Communication layer** (usually WebSocket or HTTP).

### ✅ 2. **Refactor Your Python Script**

Your script should:

* Accept parameters (e.g., repo name, issue state, labels).
* Return structured JSON (issues with fields like title, status, assignee, etc.).
* Handle authentication securely (GitHub token via environment variable or config).

Example function signature:

Python

def get\_github\_issues(repo: str, state: str = "open", labels: list = None) -> dict:

# Returns JSON with issue details

Show more lines

### ✅ 3. **Create MCP Tool Server**

You can use **FastAPI** or **Flask** for HTTP, or **WebSocket** for real-time. For MCP compliance:

* Define endpoints like /issues that call your Python function.
* Return responses in MCP-compatible JSON format.

Example FastAPI snippet:

Python

from fastapi import FastAPI

app = FastAPI()

@app.get("/issues")

def fetch\_issues(repo: str, state: str = "open"):

return get\_github\_issues(repo, state)

Show more lines

### ✅ 4. **Add MCP Manifest**

Create mcp.json:

JSON

{

"name": "github-issues-tool",

"description": "Fetches GitHub issues for a given repository",

"endpoints": [

{

"path": "/issues",

"method": "GET",

"params": {

"repo": "string",

"state": "string"

}

}

]

}

Show more lines

### ✅ 5. **Test with MCP Client**

Use an MCP client (like LangChain MCP integration or custom client) to call your tool and verify responses.

### ✅ 6. **Enhance**

* Add filtering (labels, assignee).
* Add pagination.
* Add caching for performance.
* Secure with OAuth or PAT tokens.