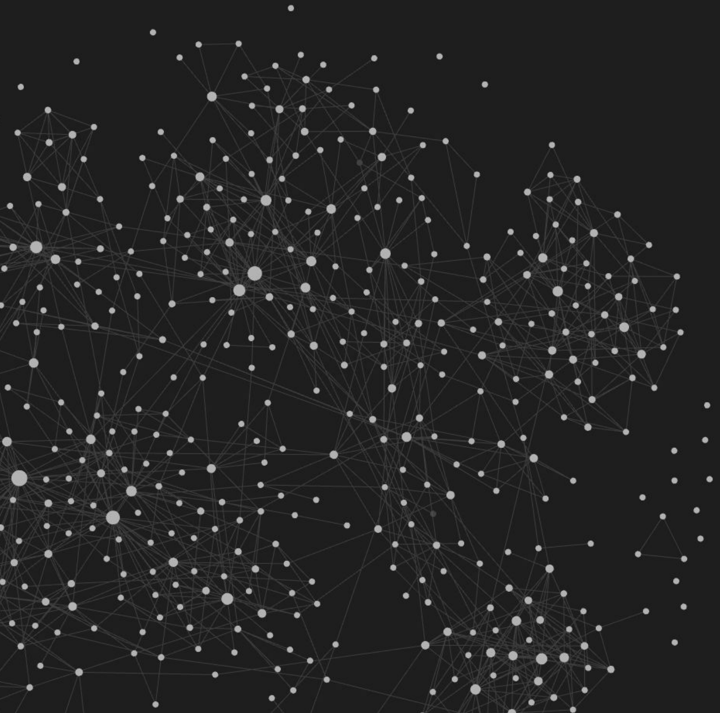
Vault Query

CS 6320

Samuel Preston

Introduction



01 What is Vault Query?

Vault Query is a plugin for Obsidian that allows users to retrieve information from their vault via RAG queries

02 What is Obsidian?

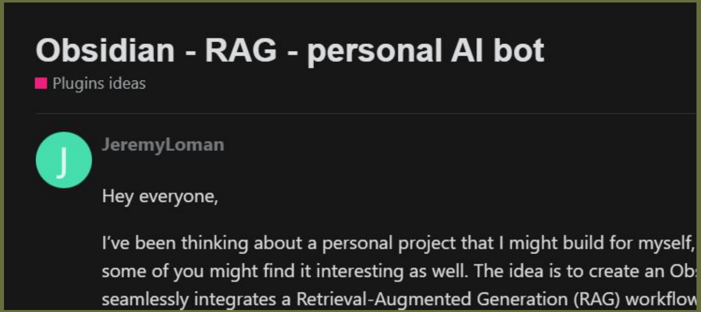
Obsidian is a popular note-taking tool that uses Markdown. Notes are stored locally in a directory called a vault.

03 Why is Vault Query useful?

Obsidian vaults are also known as “mind-maps” because they are dense network of textual information.

A chatbot would be a great tool to search for information.

Market Opportunity



Obsidian - RAG - personal AI bot

Plugins ideas

JeremyLoman

Hey everyone,

I've been thinking about a personal project that I might build for myself, some of you might find it interesting as well. The idea is to create an Ob seamlessly integrates a Retrieval-Augmented Generation (RAG) workflow

There are currently no maintained plugins for querying Obsidian vaults, but people want to.

- There are many posts on Reddit and the Obsidian forum discussing the idea of a personalized chatbot, but there are no good plugins for this.
- There was a plugin called "obsidian-rag," but it is no longer maintained.
- There are standalone programs such as "simple-rag," but these operate separately from Obsidian

Let’s Do It Ourselves

Requirements

A key selling point about Obsidian is that all data is stored locally, so:

- All information needs to be stored locally
- No external programs should be used (other than the LLM)

Obsidian is also performant, so we need to ensure:

- Startup time and general performance are not impacted
- Memory usage is limited
- Querying is fast

Obsidian is made using Electron, so:

- We need to use browser technologies (JavaScript, WASM, etc.)
- Obsidian stores persistent data using JSON

Key Technologies

Langchain.js

The JavaScript version of Langchain was used to coordinate document loading and API calls.

Voy

Voy is a vector database written in Rust and designed to be ran in the browser using WebAssembly. It is the most feature-rich browser vector database I could find.

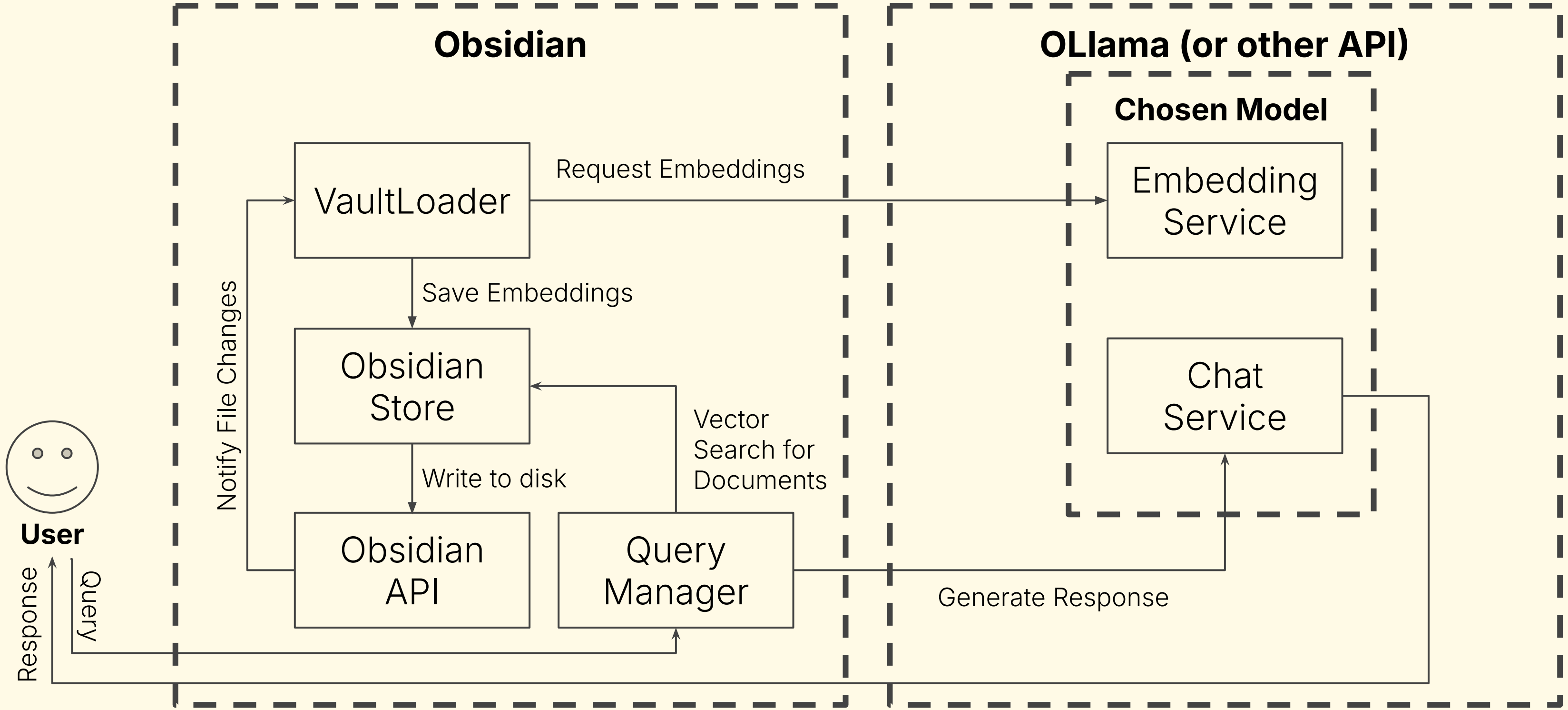
Bun

Bun is a JavaScript runtime and bundler. I used this to transpile and bundle my TypeScript code into the plugin script.

OLlama

OLlama is a program that can run LLMs locally. I used this to run Deepseek.

Architecture (Simplified)



Vault Loader

LangChain has many community document loaders, including one for Obsidian.

Unfortunately, the Obsidian loader is designed to use the NodeJS file system package. However, Obsidian does not support this.

To fix this, I wrote a conversion layer that translates all file system calls from Langchain packages into Obsidian API calls:

- `readdir => vault.adapter.list`
- `stat => vault.adapter.stat`
- `readFile => vault.getFileByPath`

To replace the file system calls, I wrote a custom Bun builder plugin that changes the module resolution path to my custom shim.

By doing this, I was able to reuse the Obsidian loader without major modifications to their source code.

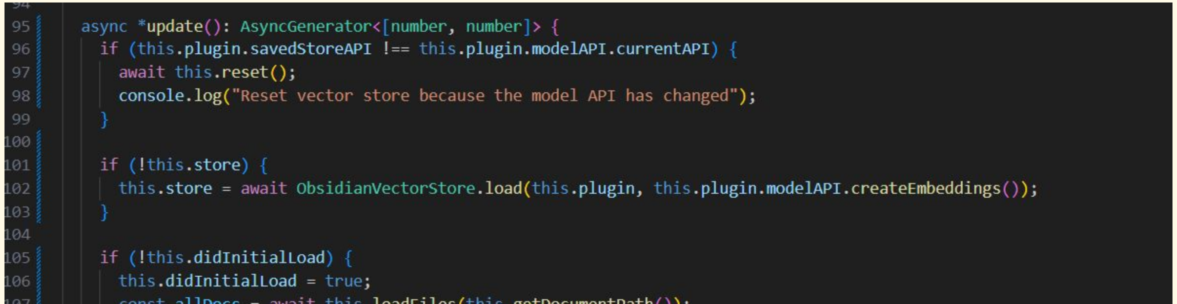
Vault Loader (cont.)

The process of loading, splitting, and embedding documents takes a long time. Furthermore, vaults are huge (mine has 600+ documents).

To keep this process manageable, I use a queue system that does the following:

- Keeps a list of un-embeded files, files to delete, and files to reload.
- Listens for file creation or modification events from Obsidian (adds to reload queue)
- Listen for file deletions or renames (adds to deletion queue)

The queue is processed using an async iterator, meaning it can be paused during execution. This allows the user to skip or pause the embedding process.



```
async *update(): AsyncGenerator<[number, number]> {
  if (this.plugin.savedStoreAPI !== this.plugin.modelAPI.currentAPI) {
    await this.reset();
    console.log("Reset vector store because the model API has changed");
  }

  if (!this.store) {
    this.store = await ObsidianVectorStore.load(this.plugin, this.plugin.modelAPI.createEmbeddings());
  }

  if (!this.didInitialLoad) {
    this.didInitialLoad = true;
    const allDocs = await this.loadFiles(this.getDocumentPath());
```

Obsidian Store

LangChain also has many supported vector stores, including Voy.

Initially, I was was going to use their implementation. However, it has many problems:

- Documents cannot be deleted once added
- The database could not be saved nor loaded

Instead, I wrote my own vector store implementation on top of Voy:

- The database can be serialized to JSON and loaded from disk using the Obsidian API.
- Individual documents can be deleted
- Entries are stored using their hash, so it is easy to identify and remove outdated documents.

By implementing the database myself, I was able to store all embedding information locally in an Obsidian-esque way.

Other Notes

In addition to the custom document processor and vector store, I did other advanced implementations:

- I wrote my own WebAssembly bundler for Bun since it currently doesn't support bundling WASM files.
- I also wrote my scripts for automatically building and activating the plugin in Obsidian to speed up development
- I used dependency injection to allow for multiple LangChain APIs to be supported such as OLama, OpenAI, and Grok.

Demo

Future Steps

Better Document Filtering

Currently, documents can only be filtered by tags, but there should also be ways to filter by folder and content.

Better Query Tools

It would be helpful if you could help the LLM find related documents by including tags in the query

PDF Support

Currently, only Markdown files are loaded. Since Obsidian supports PDFs, they should be supported too.

Publishing

This is a project that I want greatly for myself, so I want to continue to improve it and eventually publish it on the plugin hub

Code Smell

I did my best to write quality code, but I definitely need to clean it up before publishing.

Summary

Exploration Beyond Baseline

I implemented:

- A fully working Obsidian plugin
- A local-first RAG chatbot for Obsidian Markdown notes

Innovation

I implemented by own:

- LangChain vector store that runs in the browser
- Document loader that can handle dynamic file modifications

Heightened Complexity

There were many constraints because Obsidian is browser-based and limits plugin permissions.

Nevertheless, I succeeded in creating a system more complex than many standard standalone ones.