Content Library Technical Document

Intro to the AI-Based Asset Search System

This documentation outlines the steps to set up a lightweight, local AI-based asset search API using **Python**, **SQLite**, **Sentence Transformers**, **scikit-learn**, and **Flask**. It also includes integration guidance for Unity via C# web requests.

Overview

This system allows developers to:

- Store asset metadata in SQLite.
- Search using natural language.
- Match assets based on semantic similarity using Sentence Transformers.
- Filter results by asset type.
- Query the system via a RESTful Flask API.
- Integrate the API with Unity using HTTP requests.

Prerequisites

- A Windows machine
- Internet connection for downloading packages

Environment setup steps

Step 1: Install Python

1. Download and install Python from https://www.python.org/downloads/
2. During installation, check the box **"Add Python to PATH"**.
3. Verify installation:

```
python --version
```

Step 2: Install Required Python Packages

Open Command Prompt (cmd) and run:

```
pip install --upgrade pip
pip install sentence-transformers scikit-learn numpy flask
```

If you encounter errors like:

Error: "Could not install packages due to an OSError"

- Go to `C:\Python312\Scripts\`
- Delete any files like `flask.exe` or `flask.exe.deleteme`
- Re-run:
  pip install flask --force-reinstall

Error: "WARNING: Ignoring invalid distribution ~ip"

- Go to `C:\Python312\Lib\site-packages\`
- Delete broken folders like `~ip`
- Then run:

Then run:

```
python -m ensurepip --upgrade
python -m pip install --upgrade pip
```

Step 3: Create the Asset Search Engine

Save the following as `asset_search_engine.py`:

```
import sqlite3
import numpy as np
from sentence_transformers import SentenceTransformer
from sklearn.neighbors import NearestNeighbors

class AssetSearchEngine:
    def __init__(self, db_path='assets.db'):
        self.db_path = db_path
        self.model = SentenceTransformer("all-MiniLM-L6-v2")
        self.nn_model = None
        self.embeddings = []
        self.asset_ids = []
        self._load_assets_and_build_index()

    def _connect_db(self):
        return sqlite3.connect(self.db_path)

    def _load_assets_and_build_index(self):
        conn = self._connect_db()
        cursor = conn.cursor()
        cursor.execute("SELECT id, name, description FROM assets")
        rows = cursor.fetchall()
        conn.close()

        self.embeddings = []
        self.asset_ids = []
        for row in rows:
            id_, name, description = row
            self.asset_ids.append(id_)
            text = f"{name} {description}"
            embedding = self.model.encode(text)
            self.embeddings.append(embedding)

        if self.embeddings:
            n_samples = len(self.embeddings)
            n_neighbors = min(n_samples, 3)
            self.nn_model = NearestNeighbors(n_neighbors=n_neighbors,
metric='cosine')
            self.nn_model.fit(np.array(self.embeddings))

    def index_sample_assets(self, assets):
        conn = self._connect_db()
        cursor = conn.cursor()

        cursor.execute("DROP TABLE IF EXISTS assets")
        cursor.execute("""
            CREATE TABLE assets (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT,
                description TEXT,
                type TEXT,
                path TEXT
            )
        """)
        conn.commit()

        self.embeddings = []
        self.asset_ids = []

        for asset in assets:
            text = f"{asset['name']} {asset['description']}"
            vector = self.model.encode(text)
            self.embeddings.append(vector)
            cursor.execute("INSERT INTO assets (name, description, type,
path) VALUES (?, ?, ?, ?)",
                           (asset['name'], asset['description'],
asset['type'], asset['path']))
            self.asset_ids.append(cursor.lastrowid)

        conn.commit()
        conn.close()

        n_samples = len(self.embeddings)
        n_neighbors = min(n_samples, 3)
        self.nn_model = NearestNeighbors(n_neighbors=n_neighbors,
metric='cosine')
        self.nn_model.fit(np.array(self.embeddings))

    def search(self, query, asset_type=None):
        if not self.nn_model:
            return []

        query_embedding = self.model.encode(query).reshape(1, -1)
        distances, indices = self.nn_model.kneighbors(query_embedding)

        conn = self._connect_db()
        cursor = conn.cursor()

        results = []
        for idx in indices[0]:
            asset_id = self.asset_ids[idx]
            cursor.execute("SELECT path, type FROM assets WHERE id=?",
(asset_id,))
            row = cursor.fetchone()
            if row:
                path, type_ = row
                if asset_type is None or type_ == asset_type:
                    results.append(path)

        conn.close()
        return results


# Example Usage
if __name__ == "__main__":
    assets = [
        {"name": "Red Dragon Texture", "description": "A detailed texture
of a fierce red dragon.", "type": "Texture", "path":
"/assets/textures/red_dragon.png"},
        {"name": "Red Dragon Model", "description": "A 3D Model of a fierce
red dragon.", "type": "3D Model", "path":
"/assets/textures/red_dragon.fbx"},
        {"name": "Forest Background", "description": "A lush green forest
background with trees and mist.", "type": "Image", "path":
"/assets/backgrounds/forest.jpg"},
        {"name": "Magic Sword Model", "description": "A 3D model of an
enchanted sword with glowing runes.", "type": "3D Model", "path":
"/assets/models/magic_sword.obj"},
        {"name": "Player Character Prefab", "description": "A complete
player character with animations.", "type": "Prefab", "path":
"/assets/prefabs/player.prefab"},
    ]

    engine = AssetSearchEngine()
    engine.index_sample_assets(assets)

    print("Search result:", engine.search("fire dragon skin"))
    print("Search 3D:", engine.search("enchanted sword", asset_type="3D
Model"))
    print("Search Image:", engine.search("misty forest",
asset_type="Image"))
    print("Search:", engine.search("dragon character with sword"))
```

This script:

- Stores asset data in SQLite

- Computes vector embeddings using Sentence Transformers
- Builds a scikit-learn NearestNeighbors model
- Enables filtered and semantic search

Step 4: Expose a Flask API

Save the following as `search_api.py`:

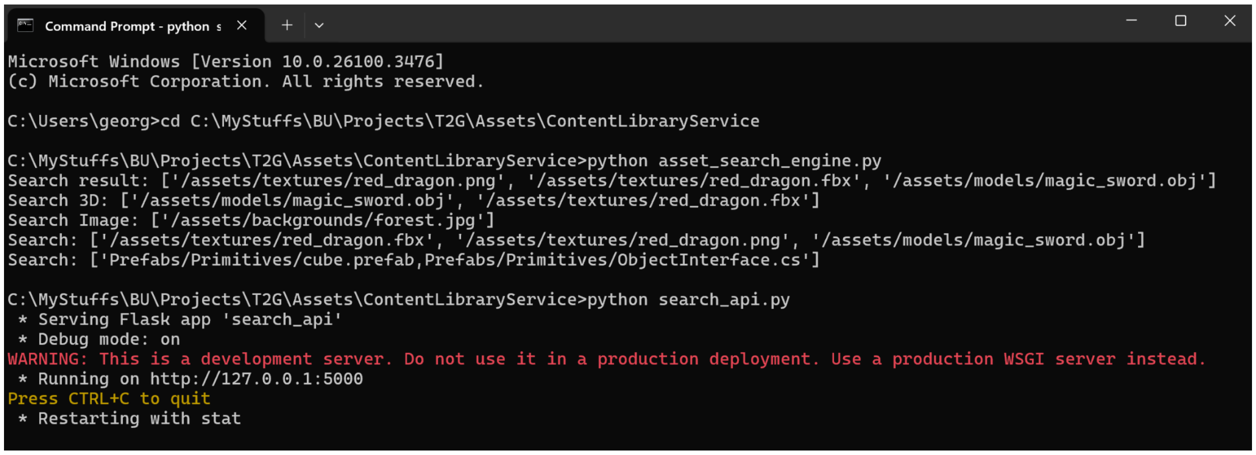
```
from flask import Flask, request, jsonify
from asset_search_engine import AssetSearchEngine

app = Flask(__name__)
engine = AssetSearchEngine()  # Load existing assets

@app.route('/search', methods=['GET'])
def search():
    query = request.args.get('q')
    asset_type = request.args.get('type', None)

    if not query:
        return jsonify({"error": "Missing query parameter ?q=..."}), 400

    results = engine.search(query, asset_type)
    return jsonify({"results": results})

if __name__ == "__main__":
    app.run(port=5000, debug=True)
```

Run it using:

```
python search_api.py
```

The API will be available at:

```
http://localhost:5000/search?q=your+query&type=optional_type
```



```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

C:\Users\georg>cd C:\MyStuffs\BU\Projects\T2G\Assets\ContentLibraryService

C:\MyStuffs\BU\Projects\T2G\Assets\ContentLibraryService>python asset_search_engine.py
Search result: ['/assets/textures/red_dragon.png', '/assets/textures/red_dragon.fbx', '/assets/models/magic_sword.obj']
Search 3D: ['/assets/models/magic_sword.obj', '/assets/textures/red_dragon.fbx']
Search Image: ['/assets/backgrounds/forest.jpg']
Search: ['/assets/textures/red_dragon.fbx', '/assets/textures/red_dragon.png', '/assets/models/magic_sword.obj']
Search: ['Prefabs/Primitives/cube.prefab,Prefabs/Primitives/ObjectInterface.cs']

C:\MyStuffs\BU\Projects\T2G\Assets\ContentLibraryService>python search_api.py
 * Serving Flask app 'search_api'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
```

Step 5: Unity Integration via C# Web Request

In Unity, use the following C# script to call the search API:

```
using UnityEngine;
using UnityEngine.Networking;
using System.Collections;

public class AssetSearchClient : MonoBehaviour
{
    public string query = "fire dragon skin";

    IEnumerator Start()
    {
        string url = "http://localhost:5000/search?q=" +
UnityWebRequest.EscapeURL(query);
        UnityWebRequest request = UnityWebRequest.Get(url);
        yield return request.SendWebRequest();

        if (request.result == UnityWebRequest.Result.Success)
        {
            Debug.Log("Response: " + request.downloadHandler.text);
        }
        else
        {
            Debug.LogError("Error: " + request.error);
        }
    }
}
```

Attach this script to a GameObject in your scene and press Play.

Conclusion

This system provides a powerful, lightweight solution for natural language asset search without relying on complex infrastructure. It is easy to extend and integrate into Unity and Unreal workflows.

For enhancements such as image previews, upload endpoints, or multi-user deployment, the Flask app can be expanded further.