**Similarity Search Project Documentation**

**Overview**

This project implements a similarity search system using PostgreSQL for data storage and retrieval, FastAPI for the API interface, and Python for backend logic. The system allows users to load Wikipedia content and perform similarity searches on the stored data.

**Technologies Used**

* PostgreSQL with pgvector extension
* Python 3.x
* FastAPI
* Postman (for API testing)
* Google's Generative AI (Gemini model)
* Windows Subsystem for Linux (WSL)

**Key Components**

1. main.py: FastAPI application entry point
2. controller.py: Handles data loading and query processing
3. llm.py: Manages database operations and similarity search logic

**Setup and Configuration:-**

**PostgreSQL Setup in WSL**

**We are using Windows Subsystem for Linux (WSL) for PostgreSQL setup due to specific requirements of the pgvector extension, which is crucial for our similarity search functionality. Here's why:**

**1. pgvector Compatibility: The pgvector extension, which adds vector similarity search capabilities to PostgreSQL, is primarily designed for and tested on Linux environments. It may not be fully compatible or easily installable on native Windows PostgreSQL installations.**

**2. Linux-like Environment: WSL provides a Linux-like environment on Windows, allowing us to leverage the full capabilities of pgvector without the need for a separate Linux machine or virtual machine.**

**3. Performance: Running PostgreSQL with pgvector in WSL can offer better performance and stability compared to attempting to run it natively on Windows or through other virtualization methods.**

**4. Ease of Development: Using WSL allows developers to work in a familiar Windows environment while having access to Linux tools and compatibility, bridging the gap between development and production environments.**

**Setup Steps:**

**1. Install WSL and a Linux distribution (e.g., Ubuntu)**

**2. Install PostgreSQL in WSL**

**3. Configure PostgreSQL:**

**\* Edit `postgresql.conf`: Set `listen\_addresses = '\*'`**

**- This allows PostgreSQL to accept connections from any IP address, necessary for connecting from the Windows host.**

**\* Edit `pg\_hba.conf`: Add host entry for Windows IP address**

**- This step is crucial for allowing your Windows host to connect to the PostgreSQL instance running in WSL.**

**\* Set authentication method (e.g., md5 or trust)**

**- Choose an appropriate authentication method based on your security requirements.**

**4. Start PostgreSQL service in WSL**

**Functionality**

**Data Loading (/load endpoint)**

1. Accepts a Wikipedia URL
2. Scrapes content (title, headings, paragraphs, lists)
3. Processes and chunks the content
4. Generates embeddings using Google's Generative AI
5. Stores data and embeddings in PostgreSQL

**Query Processing (/query endpoint)**

1. Accepts a query string
2. Generates query embedding
3. Performs similarity search in PostgreSQL
4. Retrieves relevant paragraphs
5. Generates a human-readable summary using Gemini AI

**Code Structure**

**main.py**

* Defines FastAPI application and routes
* Handles HTTP exceptions

**controller.py**

* load\_data(): Manages Wikipedia content scraping and storage
* process\_query(): Orchestrates query processing and similarity search

**llm.py**

* Database connection and operations
* Embedding generation using Google's Generative AI
* Similarity search implementation
* Summary generation using Gemini model

**Key Features**

1. Wikipedia content scraping and processing
2. Vector embeddings for efficient similarity search
3. Integration with Google's Generative AI for embedding and summary generation
4. Scalable data storage using PostgreSQL with pgvector extension

**Challenges and Solutions**

* Initially attempted to use Milvus for vector storage but faced limitations in retrieving associated content.
* Switched to PostgreSQL with pgvector extension to store both embeddings and content, enabling more comprehensive search results.

**Usage**

1. Start the FastAPI server: uvicorn main:app --reload
2. Use the /load endpoint to populate the database with Wikipedia content
3. Use the /query endpoint to perform similarity searches and retrieve relevant information

**Conclusion**

This similarity search project demonstrates the integration of modern NLP techniques with traditional database systems, providing a powerful tool for information retrieval and analysis. By leveraging PostgreSQL's vector capabilities and Google's Generative AI, the system offers efficient and insightful content discovery based on semantic similarity.