**Python Domain specific project using RAG**

This project involves developing a Coding Assistant designed specifically to provide expert support in Python programming within the EdTech domain. The assistant is built to respond to Python-related questions, focusing on beginner-friendly, clear, and structured answers.

**Key Features:**

**Specialization in Python:** The assistant exclusively handles Python programming topics. When a user asks a question, it offers a concise and clear theoretical explanation along with simple code examples, when necessary. The responses cater to learners in the EdTech space, especially for beginners.

**Structured Response Format:** The assistant follows a pre-defined response structure for each query:

**Theoretical Explanation:** A brief explanation of the Python concept in question.

**Code Snippet:** A small, relevant code snippet, if applicable, to demonstrate the concept.

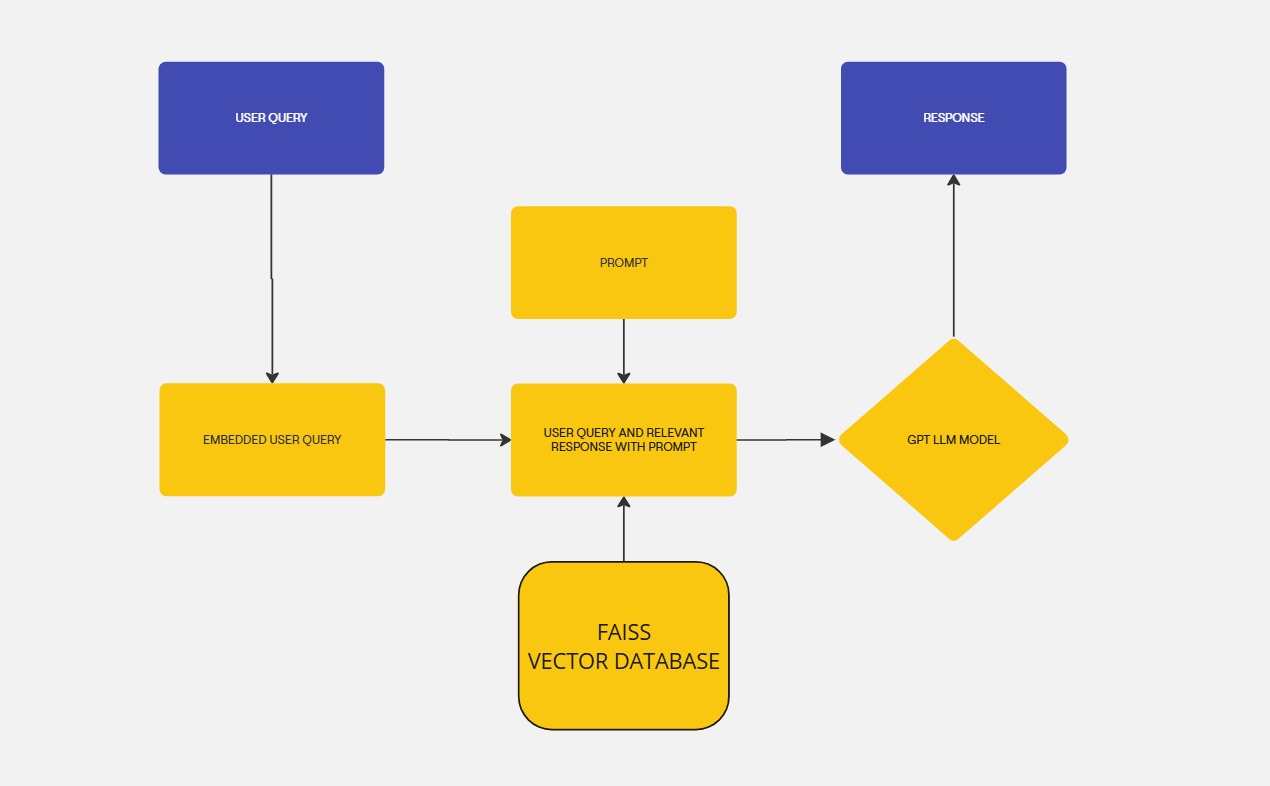
**Example:** A clear and easy-to-understand example to further illustrate the explanation.

**Polite Refusals for Non-Python Topics:** If a question pertains to another programming language or an unrelated topic, the assistant politely refuses by saying: "I only provide support for Python programming topics. Please ask something related to Python."

**Upflairs Integration:** The assistant is also equipped to answer questions about Upflairs, an EdTech company that offers courses, internships, and other learning programs. If a user’s query pertains to Upflairs, the assistant provides detailed information about the company’s offerings alongside the Python support.

**Contextual Responses:** The assistant adapts to the context of the question and provides specific answers based on what is asked. If no programming language is mentioned, it assumes Python is the subject and proceeds with an answer.

**Project Architecture**



**User Query:** A user submits a text-based query to the system.

**Prompt Creation:** The user query is transformed into a prompt, which might involve adding context, rephrasing the query, or incorporating specific instructions.

**Embedding:** The prompt is converted into a numerical representation using techniques like word embedding or sentence embedding. This creates a vector representation of the query.

**Vector Database Search:** The embedded query is stored in a FAISS vector database, which is optimized for efficiently searching and retrieving similar vectors. The database is searched for vectors that are closest to the embedded query, indicating similar meaning or context.

**Response Generation:** The most relevant vectors are retrieved from the database and used as input to a GPT LLM model. The model leverages its vast knowledge and language understanding capabilities to generate a response that is relevant to the original user query.

**Output:** The generated response is combined with the original user query and the prompt to provide a comprehensive output. This output can be presented to the user directly or used for further analysis or processing.

**Steps to execute this project**

**1. Clone the Repository**

Copy code

git clone https://github.com/delvex-community/genai.git

**2. Navigate to the project directory:**

Copy code

cd Domain\_specific\_with\_RAG

**3. Install The dependencies**

Copy code

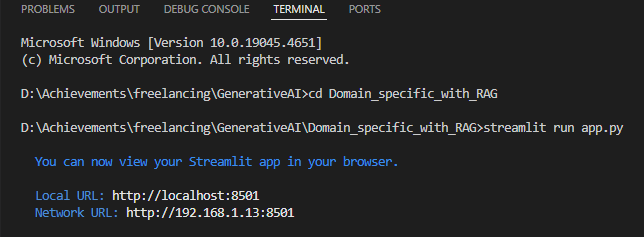
pip install -r requirements.txt

**4. Run the project file**

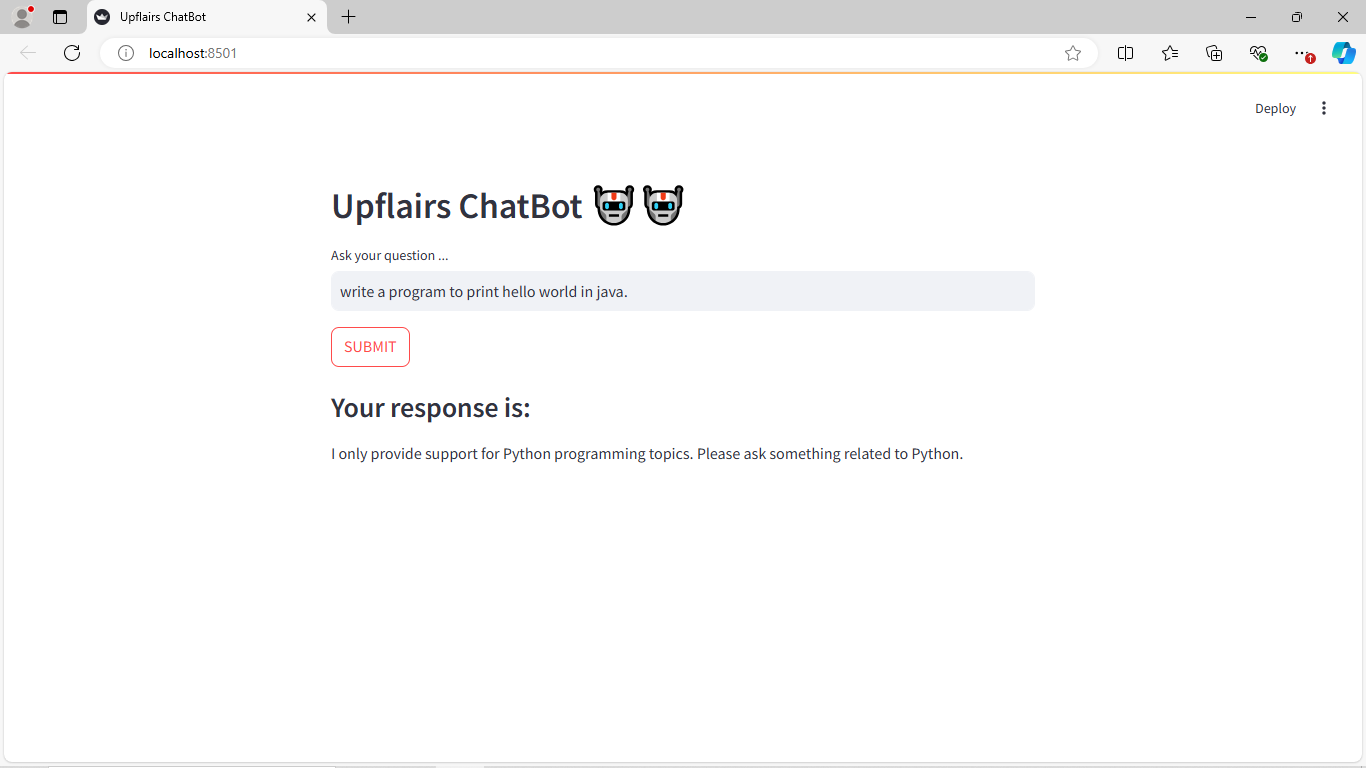
Copy code

Streamlit run app.py

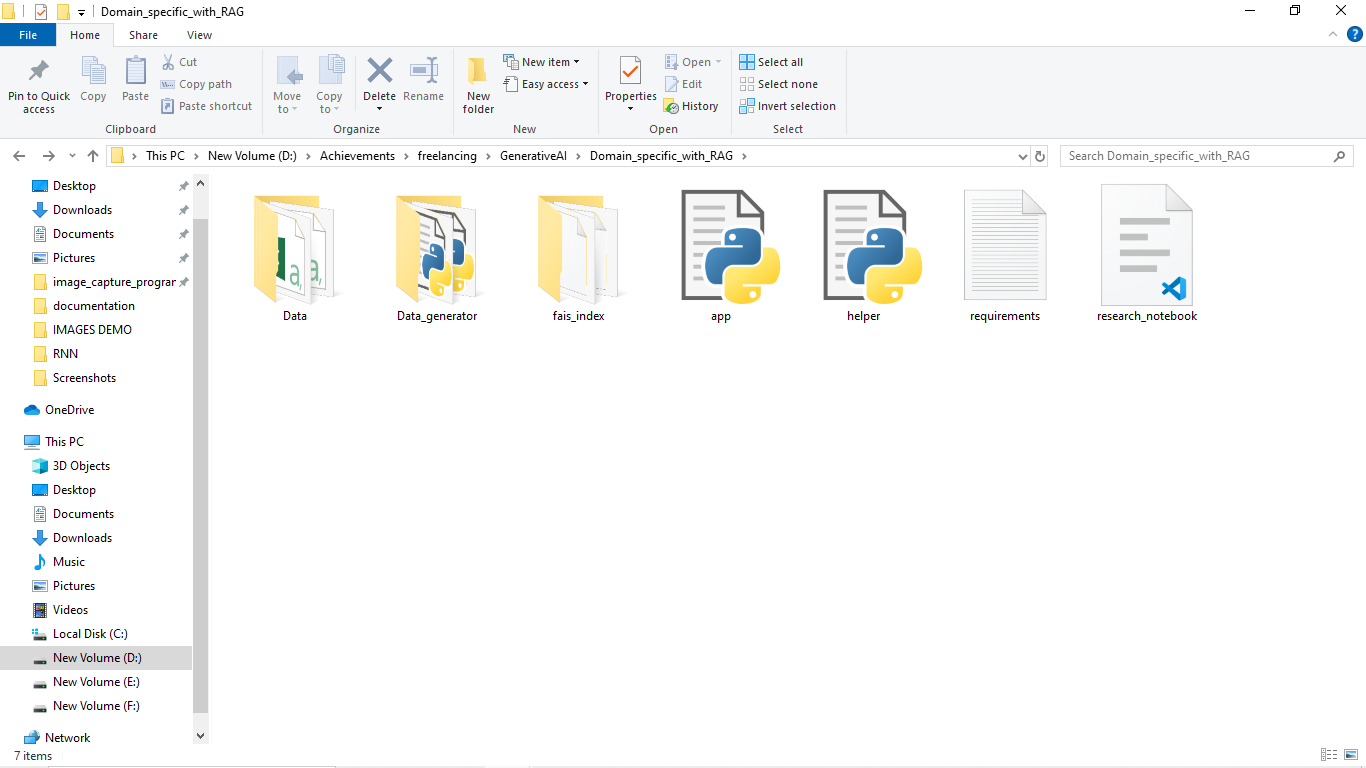
**5. Copy the local url or network url and past into browser**

****

**Application Interface**



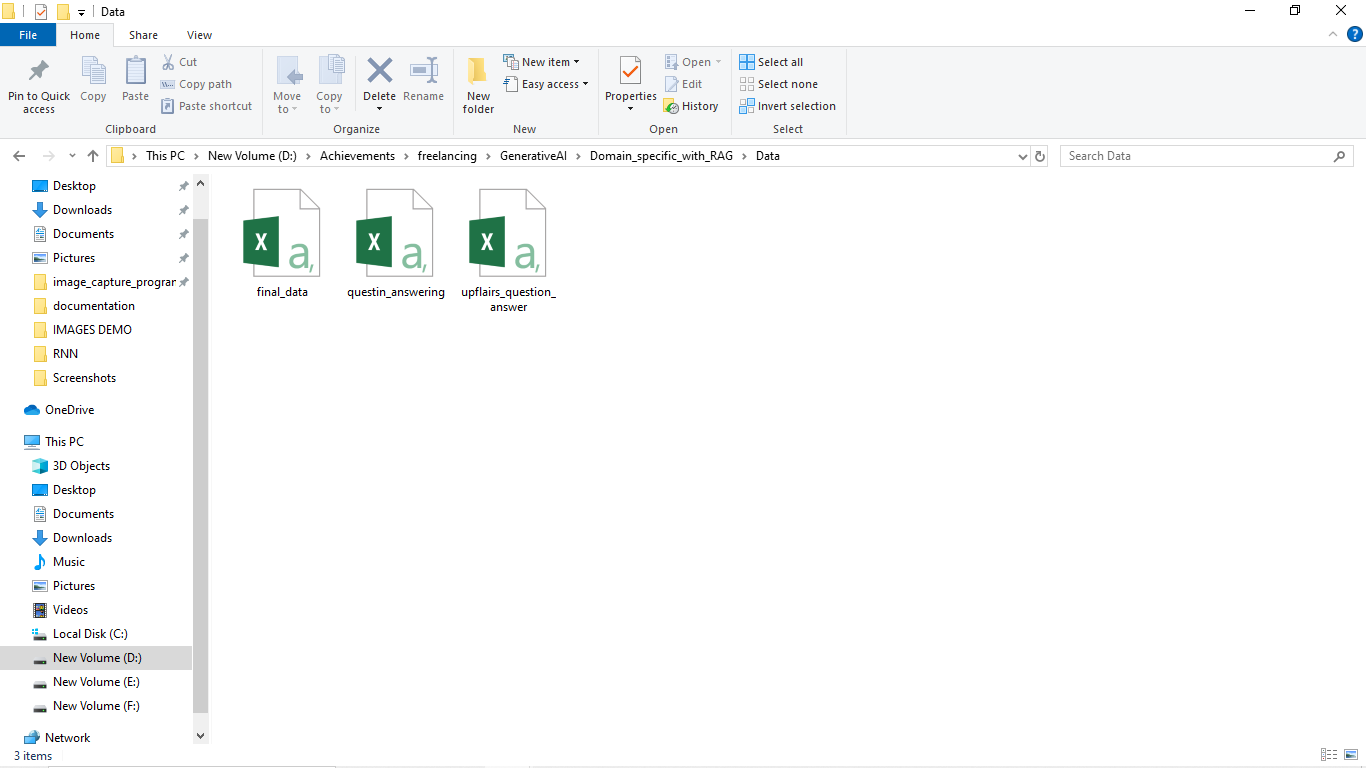
**Project folder structure**



**Brief Description of the Files and Folders**

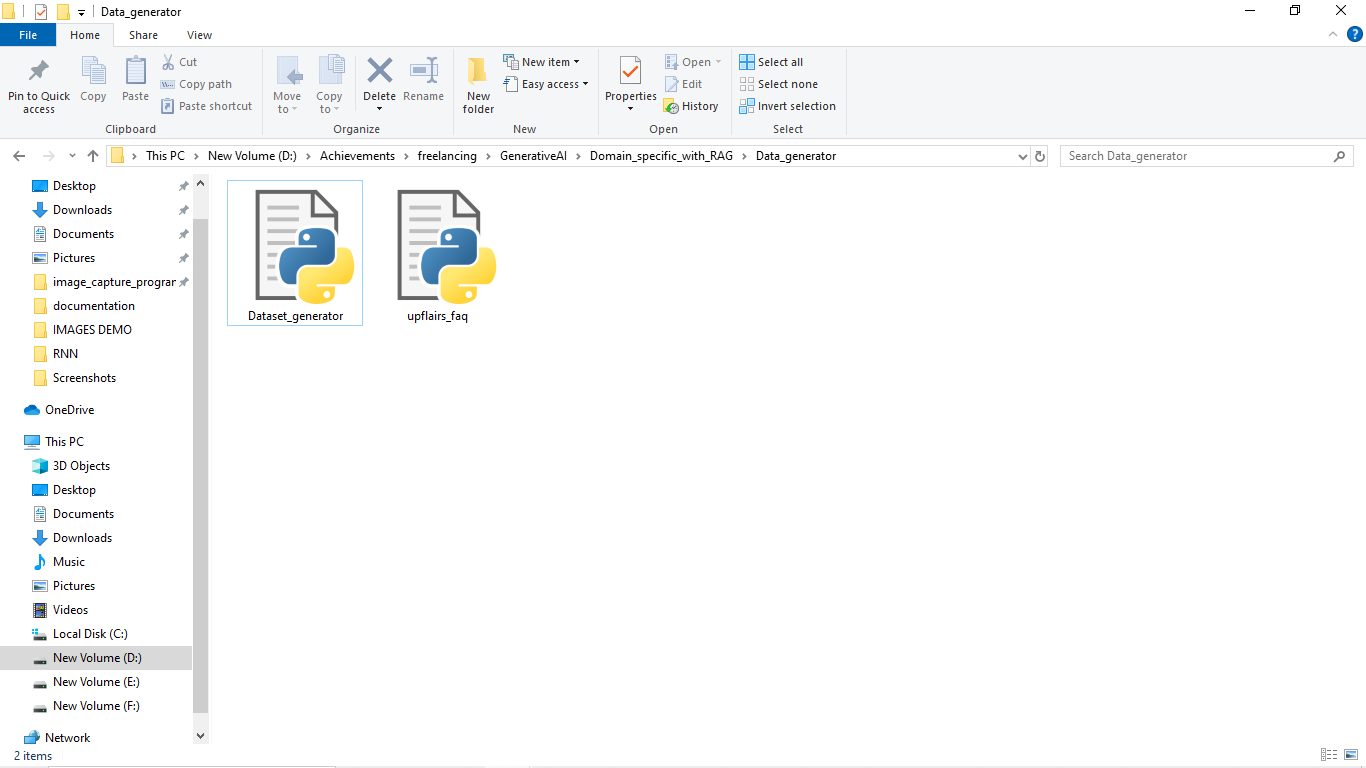
**Data:** This directory has another three “csv” files as you can see in the below image, where “**final\_data.csv**” has combined data of the two files, and files are “**questin\_answering.csv**” and “**upflairs\_question\_answer.csv**” file.

Where “**questin\_answering.csv**” has questions and answer related to python, while “**upflairs\_question\_answer.csv**” file has question and answer regarding upflairs. These files are generated by the python program files that are present inside the “**Data\_generator**”.



**Data\_generator:** it has another python files, which is responsible to generate to generate data files, that was present inside in the “data” directory,

Where “**Dataset\_generator.py**” is responsible to create “**questin\_answering.csv”** of “**data**” directory, in other hand “**upflairs\_question\_answer.csv**” is generated.



**FAISS** (Facebook AI Similarity Search) is an open-source library developed by Meta (Facebook) AI Research for efficient similarity search and clustering of dense vectors. It’s particularly popular in the fields of natural language processing (NLP) and information retrieval, where it’s used for large-scale vector search tasks like finding similar documents, images, or embeddings. Here’s a deeper look at FAISS:

Key Features of FAISS:

1. **High Performance and Scalability:**

FAISS is designed to work with extremely large datasets, often millions or even billions of vectors.

It efficiently handles high-dimensional vectors, enabling quick similarity searches even on large datasets.

1. **GPU Acceleration:**

FAISS supports GPU-based indexing and search, making it faster for large datasets and high-dimensional vectors.

1. **Clustering Capabilities:**

FAISS can be used not only for searching but also for clustering dense vectors, making it versatile for tasks requiring grouping similar embedding’s.

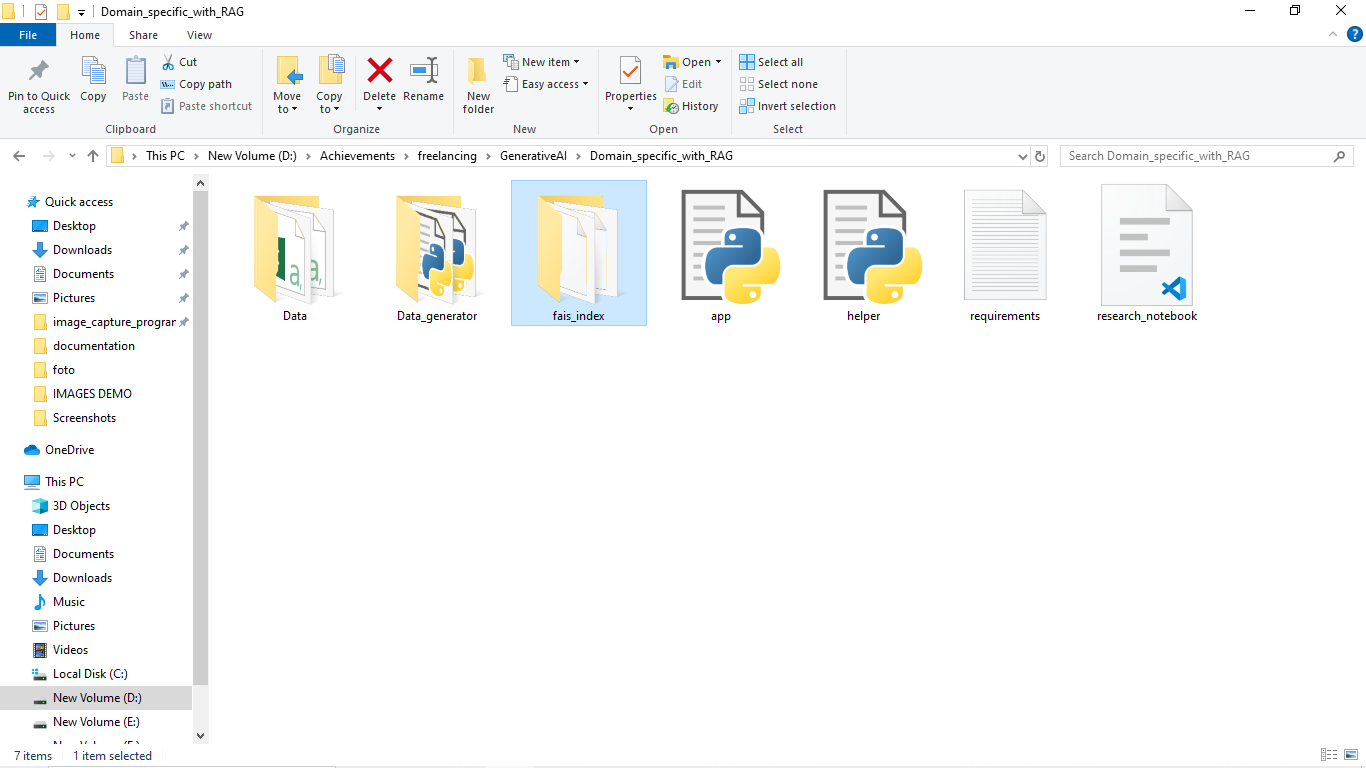
**Applications of FAISS**

* **Natural Language Processing (NLP**): Finding similar text embedding’s in tasks like document retrieval, recommendation, and question-answer matching.
* **Computer Vision:** Image similarity search, where FAISS indexes image embedding’s to enable fast visual search.
* **Recommendation Systems:** Efficiently finding similar items (e.g., products, movies) based on vectorized user or item features.
* **Content-Based Filtering:** For retrieving items that are conceptually similar to a given input, often used in personalized content suggestions.

**FAISS Works**

**FAISS** works by transforming raw data (like text, images, or other types of unstructured data) into vector embedding’s, typically through a pre-trained model or specialized embedding technique. It then indexes these vectors and, using ANN search, retrieves items based on similarity (often measured by cosine similarity or Euclidean distance). and

The **FAISS vector database** is stored locally under the name "**faiss\_index**," where I've organized Python and Upflairs-related questions and answers within the FAISS structure.



**Install dependency (requirements.txt)**

A **requirements.txt** file in a project lists all the Python dependencies needed to run the project. Each line specifies a package and its version, allowing consistent installations across environments. To install the dependencies, you use the command **pip install -r requirements.txt**. This file is essential for version control, especially in collaborative or production environments, ensuring everyone works with the same package versions.

“**requirements.txt**” file is also presented inside the main directory.

There are some other files are presented also inside the main directory,

1. **app.py**
2. **helper.py**
3. **research\_notebook.ipynb**

The **app.py** file serves as the main entry point for running the application, while **helper.py** is a separate module containing functions that support and organize the code for **app.py**. I've placed specific functions in **helper.py** to keep the codebase structured. Additionally, there's a Jupyter notebook is “**research\_notebook.ipynb**” where I experimented with various approaches to achieve my goals acting as a research notebook that documents different solutions I tried.

**Thank You 😊**