# **INTERNSHIP PROJECT REPORT**

## **Internship Title:** Infosys Springboard Virtual Internship 6.0

## **Project Title:** SwiftVisa – An Intelligent Visa Screening System Using Retrieval-Augmented Generation

# **Intern Name:** Tanisha Bablani

# **Email ID:** tanishabablani06@gmail.com

# **Internship Duration:** 2 Months

# 

# **Internship Period:** 10 November 2025 – 10 January 2026

# **Mentor Name: *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

# **Internship Mode:** Virtual Internship

# 

CHAPTER 1

**INTRODUCTION:**

In recent years, international travel has increased significantly due to globalization, educational opportunities, business expansion, and tourism. As a result, visa application processes have become more complex, country-specific, and highly rule-driven. Each country publishes detailed visa policies in the form of official documents, guidelines, and legal frameworks. These documents are often lengthy, unstructured, and written in formal legal language, making it difficult for applicants to quickly find accurate and relevant information. Many travelers rely on online blogs, agents, or informal sources, which often leads to misinformation, confusion, and incorrect visa decisions.

To address these challenges, this internship project focuses on the design and implementation of **SwiftVisa**, an intelligent Visa Screening Agent based on the **Retrieval-Augmented Generation (RAG)** architecture. Instead of relying solely on the internal knowledge of a language model, the system retrieves relevant information from official visa policy documents and uses this retrieved context to generate accurate, policy-grounded responses. This approach significantly improves the reliability, transparency, and correctness of AI-generated answers.

The SwiftVisa system integrates multiple advanced components, including document preprocessing, text chunking, semantic embeddings, vector databases, similarity-based retrieval, and large language model augmentation. Official visa documents from multiple countries are processed and stored in a structured format, enabling efficient semantic search. When a user asks a visa-related question, the system retrieves the most relevant document sections and generates a response strictly based on verified policy content.

Additionally, the system incorporates a user profile module that captures personal and travel-related details such as nationality, purpose of travel, passport validity, and previous visa rejections. This contextual information allows the system to provide more personalized and meaningful responses without violating data integrity or policy constraints. A user-friendly web interface built using Streamlit enables seamless interaction with the system, making it accessible even to non-technical users.

This internship project demonstrates how modern AI techniques can be responsibly applied to sensitive domains such as immigration and visa screening. By combining retrieval-based methods with generative models, SwiftVisa provides a scalable, accurate, and transparent solution to visa-related information retrieval. The project emphasizes correctness over creativity, ensuring that users receive trustworthy answers derived from authoritative sources.

## **1.1 Problem Statement**

Visa application processes involve strict eligibility criteria, documentation requirements, and legal conditions that vary across countries and visa categories. Applicants often struggle to understand these requirements due to the scattered nature of official information and the complexity of policy documents. Existing solutions such as search engines or chatbots either return excessive irrelevant information or provide generalized responses without policy validation.

Traditional LLM-based systems are not suitable for this domain because they may generate responses that are not directly supported by official documents. This creates a high risk of misinformation. Therefore, there is a need for an intelligent system that can retrieve accurate policy information from verified sources and generate reliable, context-aware answers.

The core problem addressed in this project is the lack of a centralized, intelligent, and trustworthy platform that can analyze official visa documents and provide accurate responses to user queries while minimizing hallucinations and ambiguity.

## **1.2 Objectives of the Project**

The primary objective of this internship project is to develop a RAG-based Visa Screening Agent that can accurately answer visa-related queries using official policy documents. The system aims to bridge the gap between complex legal documentation and user-friendly information delivery.

The specific objectives include designing an end-to-end pipeline for processing visa documents, implementing a semantic search mechanism using vector embeddings, integrating a large language model with retrieved context, and developing a web-based user interface for interaction. Another key objective is to reduce response latency while maintaining high retrieval accuracy and transparency.

**1.3 Scope of the Project**

The scope of the SwiftVisa system is limited to providing informational assistance related to visa policies, eligibility conditions, documentation requirements, and general rules. The system does not make final visa decisions, nor does it replace official government portals. Instead, it acts as a decision-support and information-retrieval tool for users

CHAPTER 2

## **2.1 Large Language Models**

Large Language Models are deep learning models trained on large-scale text data to understand and generate natural language. Models such as BERT, GPT, and Gemini use transformer architectures that rely on attention mechanisms to capture contextual relationships between words. These models are widely used for tasks such as question answering and summarization.

However, LLMs do not have direct access to verified external documents. When applied to legal or policy-driven domains, they may generate incorrect responses that sound convincing. This limitation makes standalone LLMs unreliable for visa screening applications.

## **2.2 Text Embeddings**

Text embeddings convert textual information into numerical vector representations. Semantically similar texts are placed closer together in a high-dimensional vector space. Sentence-level embeddings capture the contextual meaning of entire sentences and are highly effective for semantic search tasks.

Embedding-based retrieval has been shown to outperform keyword-based retrieval, especially for long and unstructured documents such as visa guidelines.

## **2.3 Vector Databases**

Vector databases are designed to store and retrieve high-dimensional vectors efficiently. They enable similarity-based search using distance metrics such as cosine similarity or Euclidean distance. FAISS is a widely used vector database that supports fast nearest-neighbor search.

### **Table : Comparison of Existing Approaches**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Approach** | **Data Source** | **Accuracy** | **Hallucination Risk** | **Suitability for Visa Policies** |
| Search Engines | Web pages | Medium | High | Low |
| Keyword-based IR | Documents | Medium | None | Medium |
| Standalone LLM | Model knowledge | Low | Very High | Very Low |
| RAG-based System (SwiftVisa) | Official PDFs | High | Very Low | High |

## 

## 

## 

## 

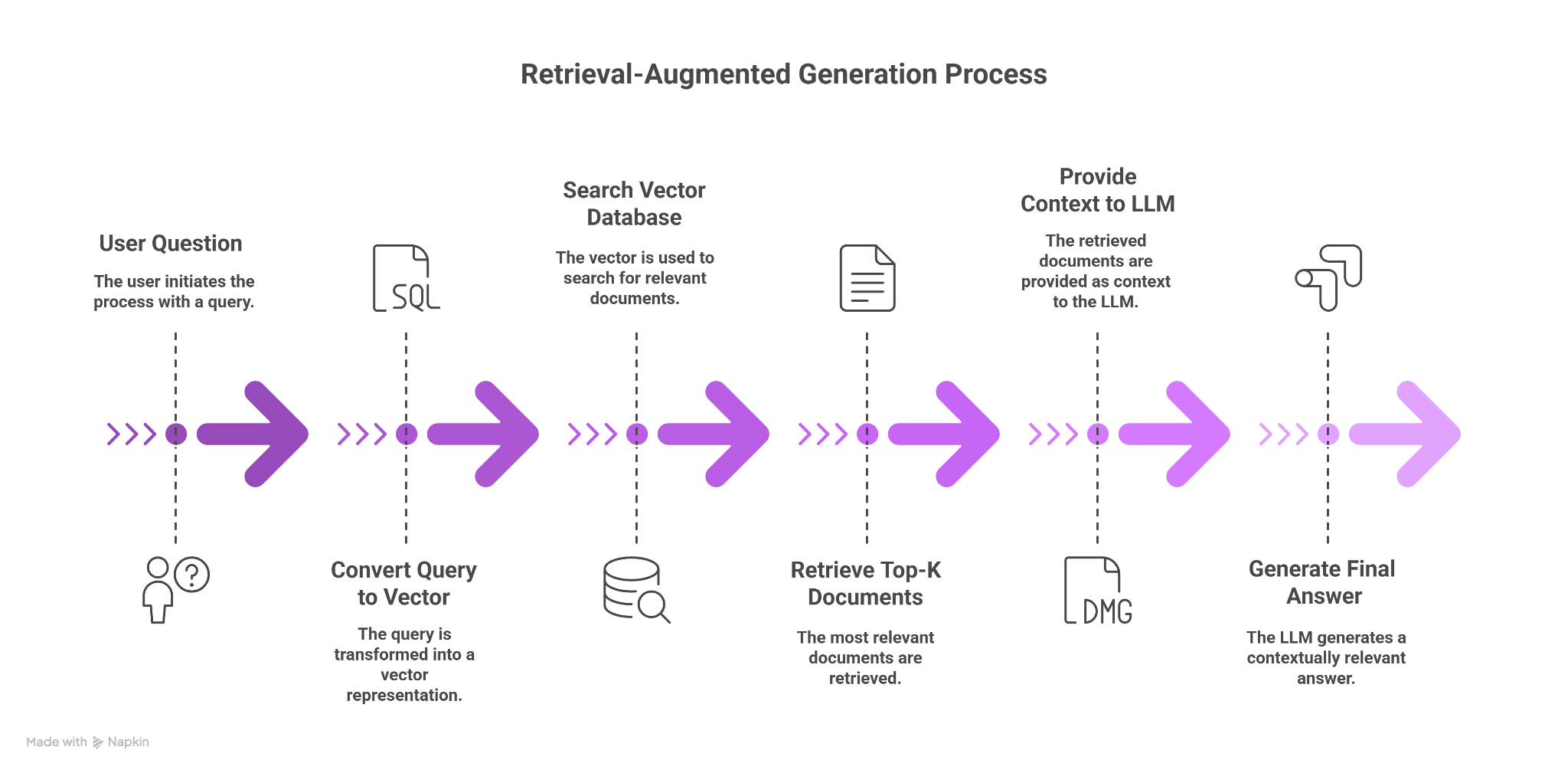
## 

## 

## **2.4 Retrieval-Augmented Generation (RAG)**

Retrieval-Augmented Generation combines document retrieval with language model generation. The system first retrieves relevant document chunks and then uses these chunks as context for response generation.

### **Flowchart : Conceptual RAG Flow**



CHAPTER 3

**SYSTEM ARCHITECTURE, PIPELINE, AND ALGORITHM**

## **3.1 System Overview**

SwiftVisa is designed as an end-to-end Retrieval-Augmented Generation system for visa screening and policy analysis. The architecture follows a modular design that separates document processing, retrieval, generation, and user interaction. This modularity ensures scalability, maintainability, and transparency.

## **3.2 High-Level System Architecture**

### **Overall System Architecture of SwiftVisa**

The pipeline begins with official visa documents in PDF format, which are processed through a document ingestion module. Extracted text is segmented into chunks and converted into embeddings. These embeddings are stored in a vector database. When a user submits a query via the Streamlit interface, the system retrieves relevant chunks and passes them to the language model for response generation.

## **3.3 Document Processing and Chunking**

Document processing involves converting unstructured PDF documents into clean textual data. Since visa documents are lengthy, they are divided into overlapping chunks to preserve context across boundaries. Each chunk is associated with metadata such as source document and chunk identifier.

The extracted text is divided into fixed-size segments with partial overlap. This approach ensures that important policy statements spanning multiple sections are not lost during segmentation.

**3.4 Embedding and Vector Storage**

Each text chunk is transformed into a numerical embedding using a sentence-level embedding model. These embeddings represent the semantic meaning of the text and are stored in a FAISS-based vector database for efficient similarity search.

## **3.5 Retrieval and Ranking Mechanism**

When a user submits a query, the query is embedded using the same embedding model. The system performs similarity search against the vector database and retrieves the top-k most relevant chunks based on distance metrics.

## 

## **3.6 Retrieval-Augmented Generation Pipeline**

Retrieved document chunks are combined with the user query and user profile information to form a structured prompt. The language model generates a response strictly based on this context, ensuring policy compliance.

## **3.7 Pseudocode**

### **Pseudocode:**

Algorithm SwiftVisa\_RAG(Q, D, U):

1. Load visa documents D

2. Extract text from PDFs

3. Split text into overlapping chunks

4. Generate embeddings for each chunk

5. Store embeddings in vector database

6. Receive user query Q

7. Generate embedding for Q

8. Perform similarity search in vector database

9. Retrieve top-k relevant chunks

10. Build prompt using:

- Retrieved chunks

- User profile U

- Strict instructions

11. Send prompt to language model

12. Generate response A

13. If no relevant data found:

Return controlled fallback response

14. Display A via Streamlit UI

CHAPTER 4

**DATASET DESCRIPTION AND PREPROCESSING**

## **4.1 Introduction**

The effectiveness of a Retrieval-Augmented Generation (RAG) system heavily depends on the quality and structure of the dataset used for retrieval. Since the SwiftVisa system is designed to answer visa-related queries accurately, it relies on official visa policy documents as its primary knowledge source. These documents contain authoritative information regarding visa eligibility, documentation requirements, restrictions, and procedural guidelines.

However, official visa documents are usually available in PDF format and are not structured for direct use in machine learning or retrieval systems. Therefore, a systematic preprocessing pipeline is required to convert raw documents into a format suitable for semantic search and retrieval. This chapter describes the dataset used in the SwiftVisa project and explains the preprocessing steps applied before embedding and indexing.

## **4.2 Dataset Description**

The dataset used in this project consists of a curated collection of official visa policy documents obtained from government and embassy portals. These documents include visitor visa guidelines, immigration rules, eligibility criteria, and application instructions. The dataset was selected to represent different countries and policy structures while ensuring reliability and authenticity.

The documents vary significantly in length, format, and complexity. Some documents are concise guidelines, while others span several pages and include legal terminology. This diversity makes the dataset suitable for evaluating the robustness of the RAG-based system.

### **Table : Overview of Visa Policy Dataset**

|  |  |  |  |
| --- | --- | --- | --- |
| **Document Type** | **Source** | **Format** | **Approximate Length** |
| Visitor Visa Guidelines | Government Portals | PDF | 10–30 pages |
| Immigration Rules | Official Immigration Websites | PDF | 40–70 pages |
| Embassy Information Sheets | Embassy Websites | PDF | 5–15 pages |
| Application Instructions | Official Visa Authorities | PDF | 15–25 pages |

This dataset structure ensures that the system is exposed to both short and long documents, simulating real-world usage scenarios.

## **4.3 Need for Preprocessing**

Raw PDF documents are not directly suitable for semantic retrieval. PDFs often contain formatting elements such as headers, footers, page numbers, and multi-column layouts that interfere with text processing. Additionally, long documents cannot be embedded as a whole due to model input limitations and retrieval inefficiency.

Preprocessing is therefore required to:

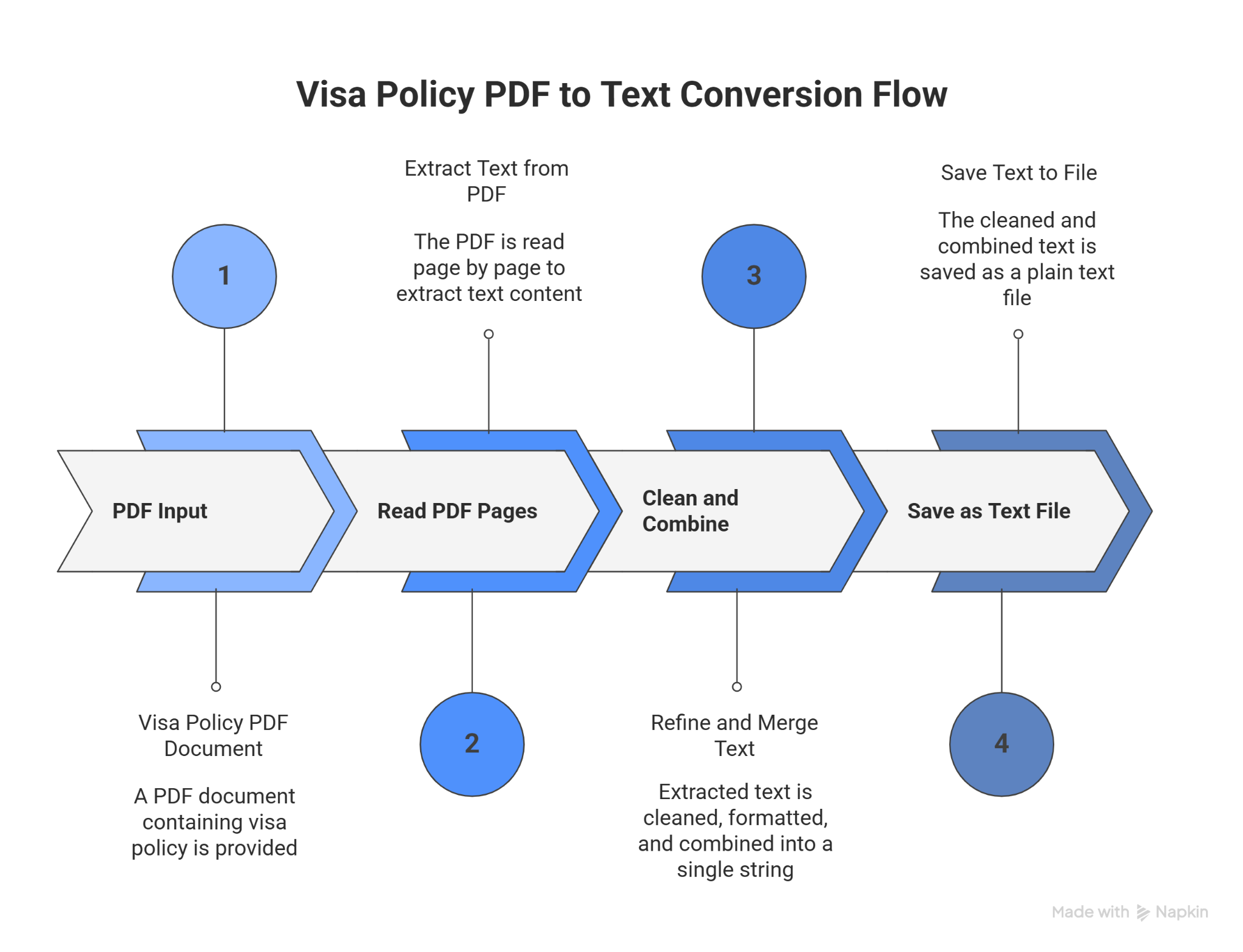
* Convert unstructured PDFs into clean textual data
* Remove formatting artifacts
* Enable efficient chunking and embedding
* Preserve the semantic meaning of policy content

## **4.4 PDF Text Extraction**

The first preprocessing step involves extracting text from PDF files. Each visa document is processed page by page, and textual content is extracted while ignoring non-textual elements such as images and formatting symbols. The extracted text is saved as plain text files for further processing.

This step standardizes the dataset and enables uniform handling of documents irrespective of their original formatting.

### **Flowchart : PDF Text Extraction Process**



## **4.5 Text Cleaning and Normalization**

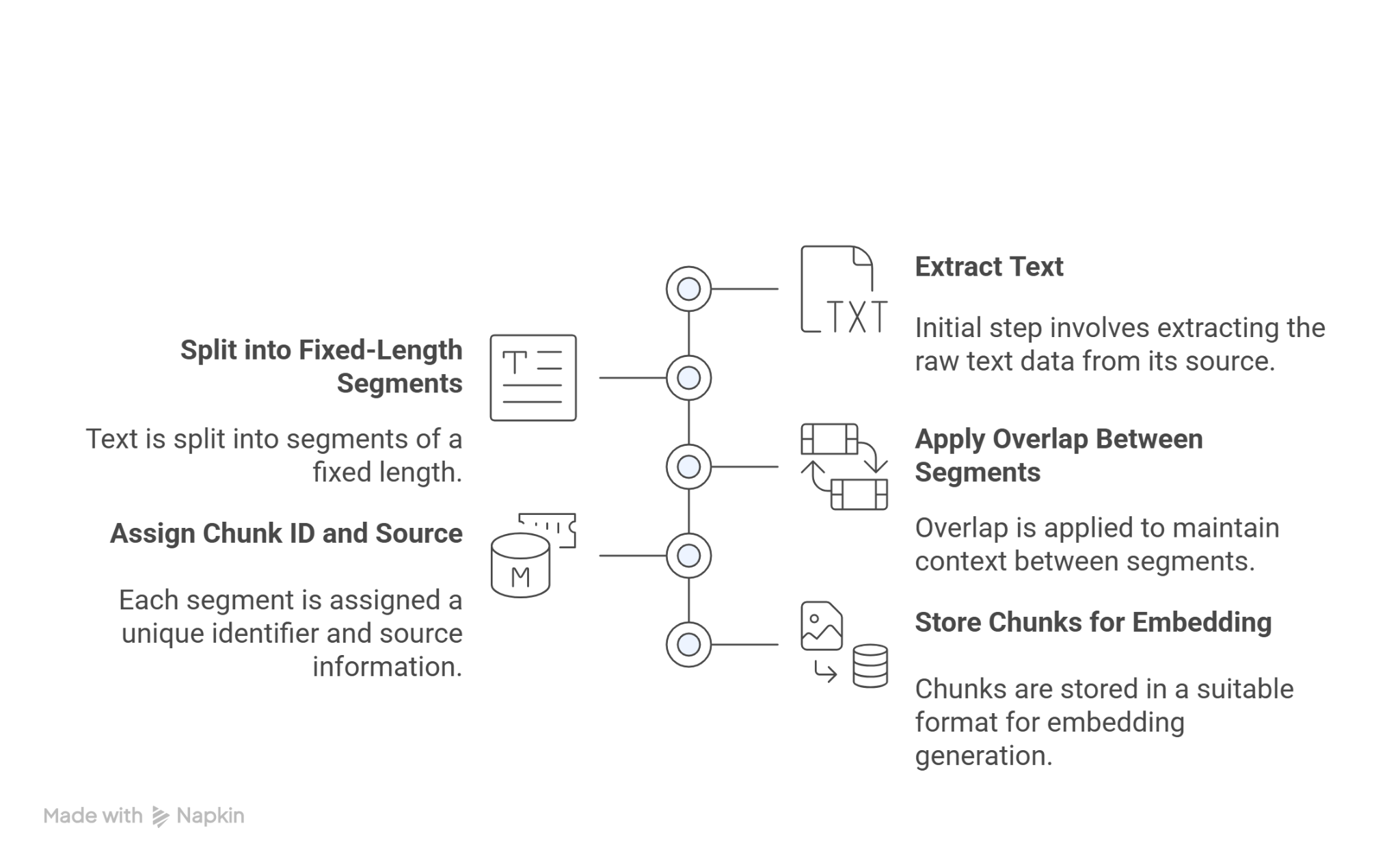
After extraction, the raw text may contain unwanted characters, excessive whitespace, and broken sentences. Text cleaning ensures that the content is readable and suitable for chunking. Normalization helps maintain consistency across documents originating from different sources.

Due to the large size of visa documents, the extracted text is divided into smaller segments known as chunks. Chunking allows the system to retrieve only the most relevant sections of a document instead of processing the entire file. This significantly improves both accuracy and efficiency.

### **Table : Chunking Parameters Used**

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Value** | **Description** |
| Chunk Size | 700 characters | Length of each text chunk |
| Overlap Size | 100 characters | Shared text between chunks |
| Chunk Type | Sliding Window | Maintains context continuity |

### **Flowchart : Text Chunking Process**



## 

## **4.6 Chunk Metadata Structure**

Each chunk generated during preprocessing contains additional metadata to improve system interpretability. This metadata helps identify the source document and track which section of the document was used to generate a response.

### **Table : Chunk Metadata Fields**

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| Chunk ID | Unique identifier for each chunk |
| Chunk Text | Actual content of the chunk |
| Source File | Original document name |

## **4.7 Prepared Dataset for Embedding**

After chunking, the dataset is transformed into a structured collection of text segments with associated metadata. This processed dataset forms the final input for the embedding generation stage. By this stage, the dataset is free from formatting noise, uniformly structured, and optimized for semantic representation.

CHAPTER 5

**METHODOLOGY: EMBEDDINGS, VECTOR DATABASE, AND RETRIEVA**L

## **5.1 Introduction**

This chapter explains the core methodology used in the SwiftVisa system to enable accurate and efficient retrieval of visa policy information. After preprocessing the dataset into structured text chunks, the next step is to convert these chunks into numerical representations that can be searched semantically. This is achieved using text embeddings, vector databases, and similarity-based retrieval techniques.

## **5.2 Text Embeddings**

Text embeddings are numerical vector representations that capture the semantic meaning of textual content. Instead of relying on exact keyword matching, embeddings allow the system to compare the meaning of a user query with the meaning of document chunks. Semantically similar texts are represented by vectors that are closer to each other in high-dimensional space.

In the SwiftVisa system, sentence-level embeddings are used because visa-related information often spans complete sentences or paragraphs. Sentence-level embeddings provide better semantic understanding compared to word-level embeddings, especially for legal and policy documents.

### **Table : Embedding Model Configuration**

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Embedding Model | SentenceTransformer |
| Model Name | all-MiniLM-L6-v2 |
| Vector Dimension | 384 |
| Input Type | Sentence / Paragraph |
| Output Type | Dense Vector |

## 

## 

## 

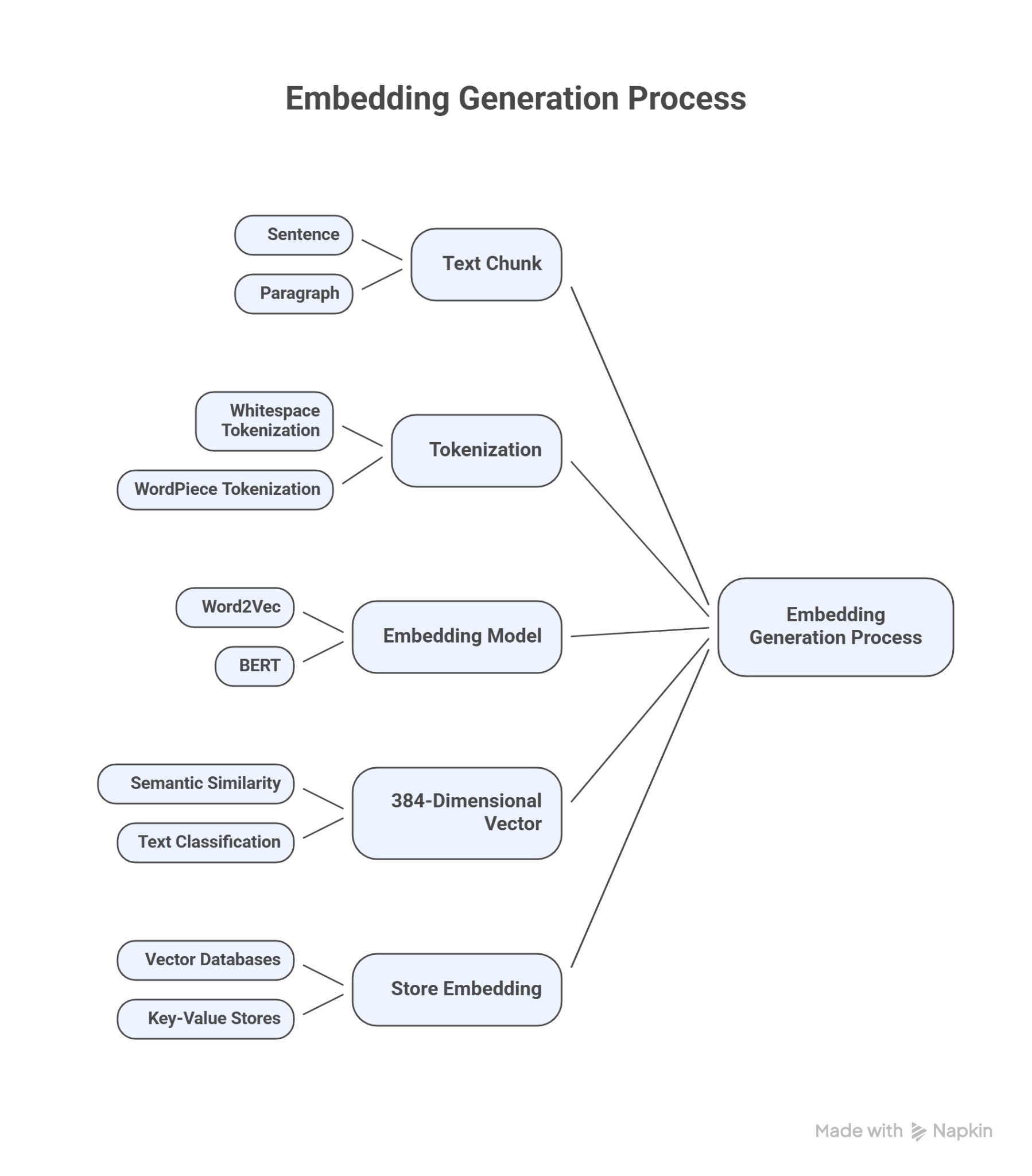
## 

## 

## **5.3 Embedding Generation Process**

Each chunk produced during preprocessing is passed through the embedding model to generate a fixed-length numerical vector. The same embedding model is used for both document chunks and user queries to ensure consistency in the semantic space.

Embedding generation is performed in batches to improve computational efficiency. The resulting embeddings are stored in a structured format for indexing and retrieval.

**Flowchart : Embedding Generation Workflow**

## **5.4 Vector Database**

A vector database is used to store and manage the embeddings generated from the document chunks. Unlike traditional databases, vector databases are optimized for similarity search in high-dimensional spaces. This allows the system to efficiently retrieve relevant document chunks based on semantic similarity.

The SwiftVisa system uses FAISS (Facebook AI Similarity Search) as the vector database. FAISS supports fast nearest-neighbor search and is suitable for large-scale embedding storage.

## **5.5 Similarity Measurement**

Similarity measurement determines how closely a user query matches stored document chunks. The SwiftVisa system uses distance-based similarity metrics to rank document chunks. Lower distance values indicate higher semantic similarity.

The system retrieves the top-k most similar chunks for each query. These chunks are considered the most relevant policy sections and are passed to the generation stage.

## **5.6 Integration with RAG Pipeline**

The retrieved document chunks form the contextual input for the Retrieval-Augmented Generation pipeline. These chunks are combined with the user query and user profile information to construct a structured prompt. The language model generates responses strictly based on this retrieved context.

This approach ensures that the system does not rely on unsupported assumptions and provides transparent, policy-grounded answers.

## **5.8 Advantages of the Proposed Methodology**

The methodology used in SwiftVisa offers several advantages:

* Accurate semantic retrieval of policy information
* Reduced hallucination risk
* Scalability to large document collections
* Fast response times due to efficient indexing
* Clear traceability of information sources

### **Pseudocode**

1. Load preprocessed text chunks

2. Initialize embedding model

3. For each text chunk:

a. Generate embedding vector

b. Store embedding in vector database

4. Receive user query

5. Generate embedding for user query

6. For each stored document embedding:

a. Compute similarity distance with query embedding

7. Rank document chunks based on similarity scores

8. Select top-K most relevant chunks

9. Return retrieved document chunks

CHAPTER 6

**PROMPT ENGINEERING AND LARGE LANGUAGE MODEL INTEGRATION**

**6.1 Introduction**

Prompt engineering plays a crucial role in Retrieval-Augmented Generation (RAG) systems. Even when accurate document chunks are retrieved, the quality of the final response depends heavily on how the information is presented to the Large Language Model (LLM). Poorly structured prompts can lead to vague, incomplete, or hallucinated responses.

In the SwiftVisa system, prompt engineering is carefully designed to ensure that the LLM generates answers strictly based on official visa policy documents. This chapter explains how prompts are constructed, how user profile data is incorporated, and how the language model is controlled to minimize hallucination.

## **6.2 Prompt Structure in SwiftVisa**

The prompt used in SwiftVisa is structured into clearly defined sections. This structure ensures that the LLM understands the context, the task, and the constraints before generating a response.

### **Table : Prompt Components in SwiftVisa**

|  |  |
| --- | --- |
| **Prompt Section** | **Description** |
| System Role | Defines the AI as a visa policy assistant |
| User Profile | Includes nationality, purpose, passport details |
| Retrieved Context | Official visa policy chunks |
| Instructions | Rules to avoid hallucination |
| User Question | Original user query |

## **6.3 User Profile Injection into Prompts**

The SwiftVisa system incorporates user-specific information to provide contextualized explanations. The user profile does not determine visa approval but helps tailor explanations according to the applicant’s situation.

User profile fields include:

* Nationality
* Country of residence
* Travel purpose
* Passport expiry date

**6.4 Large Language Model Integration**

The SwiftVisa system integrates a Large Language Model through an API-based approach. The model is used only at the final stage of the pipeline, after relevant policy information has been retrieved. This ensures that the LLM acts as an interpreter of policy documents rather than a knowledge source.

The LLM receives:

* Structured prompt
* Retrieved policy chunks
* User profile context

The response generated by the model is returned to the user through the web interface.

## **6.5 Pseudocode for Prompt Engineering**

### **Pseudocode**

1. Receive user query

2. Retrieve relevant document chunks

3. Load user profile data

4. If chunks are available:

a. Create strict prompt

b. Restrict model to retrieved content

Else:

a. Create flexible prompt

b. Allow general explanation

5. Append:

- User profile information

- Retrieved policy context

- System instructions

- User query

6. Send prompt to language model

7. Generate response

8. Return response to user interface

## **6.6 Advantages of the Prompt Engineering Approach**

The prompt engineering strategy used in SwiftVisa offers the following benefits:

* Minimizes hallucination
* Ensures policy-grounded responses
* Supports user-specific contextual explanations
* Improves transparency and reliability
* Enhances user trust

CHAPTER 7

**USER PROFILE MODULE AND PERSONALIZATION**

## **7.1 Introduction**

Personalization plays an important role in providing meaningful and relevant visa-related explanations. Visa requirements often depend on factors such as nationality, purpose of travel, duration of stay, and passport validity. A generic response may not fully address the concerns of individual applicants. Therefore, the SwiftVisa system includes a dedicated User Profile Module to capture user-specific information and incorporate it into the response generation process.

## **7.2 Purpose of the User Profile Module**

The primary purpose of the user profile module is to provide contextual information to the system without making eligibility decisions. The system does not approve or reject visas; instead, it uses profile data to tailor explanations according to the applicant’s situation.

For example, passport validity requirements or visa rules may differ depending on nationality or travel purpose. By including such information, the system can generate more precise and informative responses.

## **7.3 User Profile Information Collected**

The SwiftVisa system collects only essential information required for contextualization. The profile data is entered by the user through a structured form in the web interface.

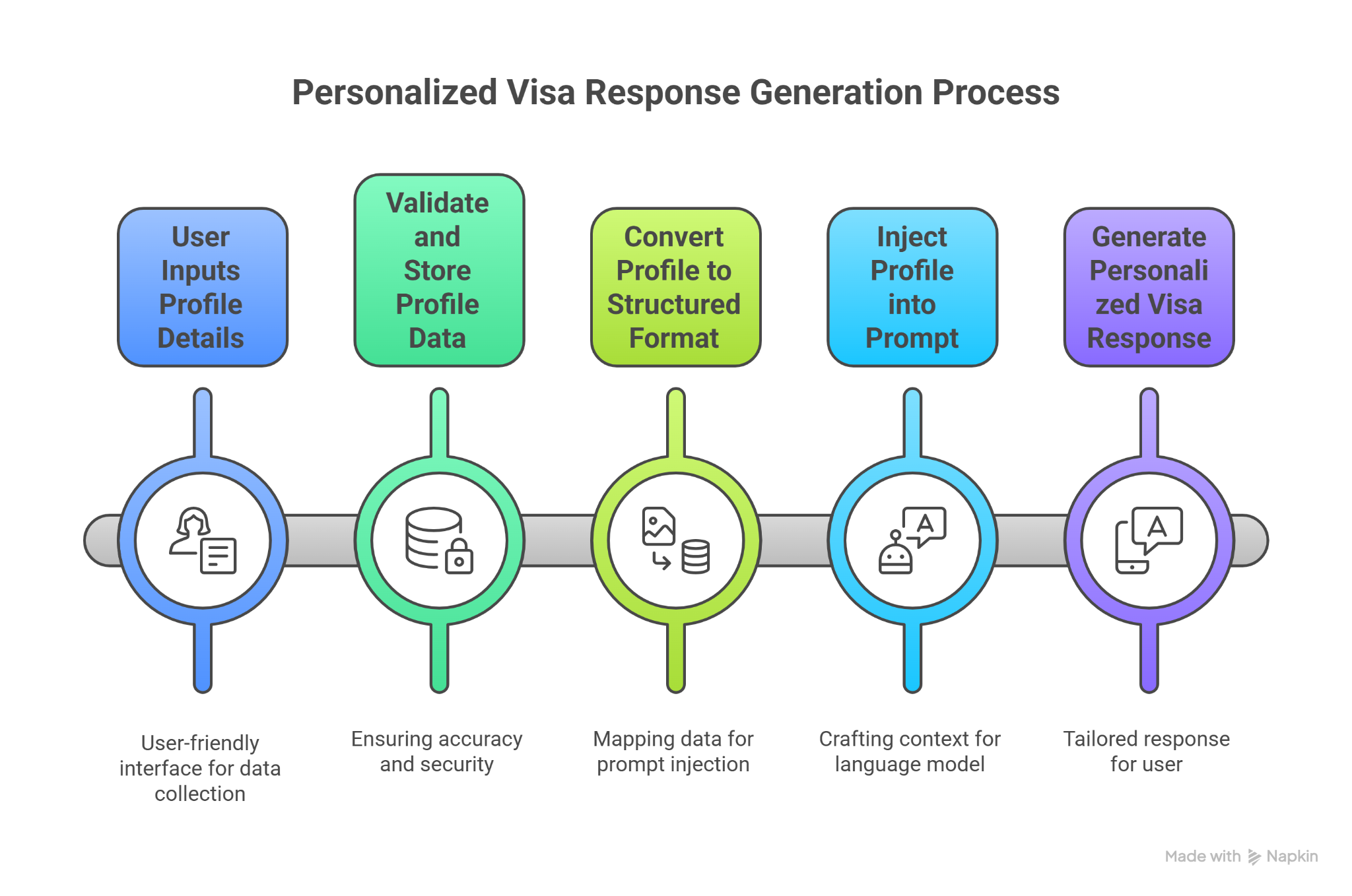
### **Table : User Profile Fields**

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| Full Name | Name of the applicant |
| Age | Applicant’s age |
| Nationality | Country of citizenship |
| Country of Residence | Current country of residence |
| Passport Expiry Date | Validity of passport |
| Purpose of Travel | Tourism, Study, Work, etc. |
| Duration of Stay | Intended length of stay |
| Travel Date | Planned travel date |
| Employment Status | Student, Salaried, Self-employed |
| Visa Rejection History | Previous rejection details (if any) |

## **7.4 User Profile Data Flow**

User profile data follows a structured flow from input to response generation. Once entered, the data is stored temporarily and injected into the prompt used by the language model.

### **Flowchart: User Profile Data Flow**



## **7.5 Integration of User Profile with RAG Pipeline**

The user profile module is integrated with the Retrieval-Augmented Generation pipeline at the prompt construction stage. Retrieved visa policy chunks remain the primary source of truth, while user profile data acts as contextual guidance.

The system ensures that profile data does not override official policy content. Instead, it helps interpret the policy in a user-specific context.

### **Pseudocode**

1. Display user profile form

2. Accept profile input values

3. Validate mandatory fields

4. Store profile data in session state

5. Convert profile data to structured format

6. Attach profile data to prompt

7. Pass prompt to RAG pipeline

CHAPTER 8

**USER INTERFACE AND SYSTEM IMPLEMENTATION (STREAMLIT)**

## **8.1 Introduction**

A user-friendly interface is essential for the practical usability of any intelligent system. While the backend of SwiftVisa focuses on accurate document retrieval and response generation, the frontend ensures that users can interact with the system easily and intuitively. The SwiftVisa user interface is developed using the Streamlit framework, which allows rapid development of interactive web applications using Python.

This chapter explains the design, components, and workflow of the user interface, along with how it integrates with the backend Retrieval-Augmented Generation pipeline.

## **8.2 Choice of Streamlit Framework**

Streamlit was selected for the implementation of the SwiftVisa interface due to its simplicity, flexibility, and seamless integration with Python-based machine learning workflows. Streamlit enables real-time interaction, dynamic form handling, and efficient visualization without requiring extensive frontend development.

The framework also supports modular page-based navigation, which aligns well with the multi-component design of the SwiftVisa system.

## **8.3 User Interface Modules**

The SwiftVisa interface is divided into multiple modules to improve usability and organization. Each module is designed to serve a specific function within the system.

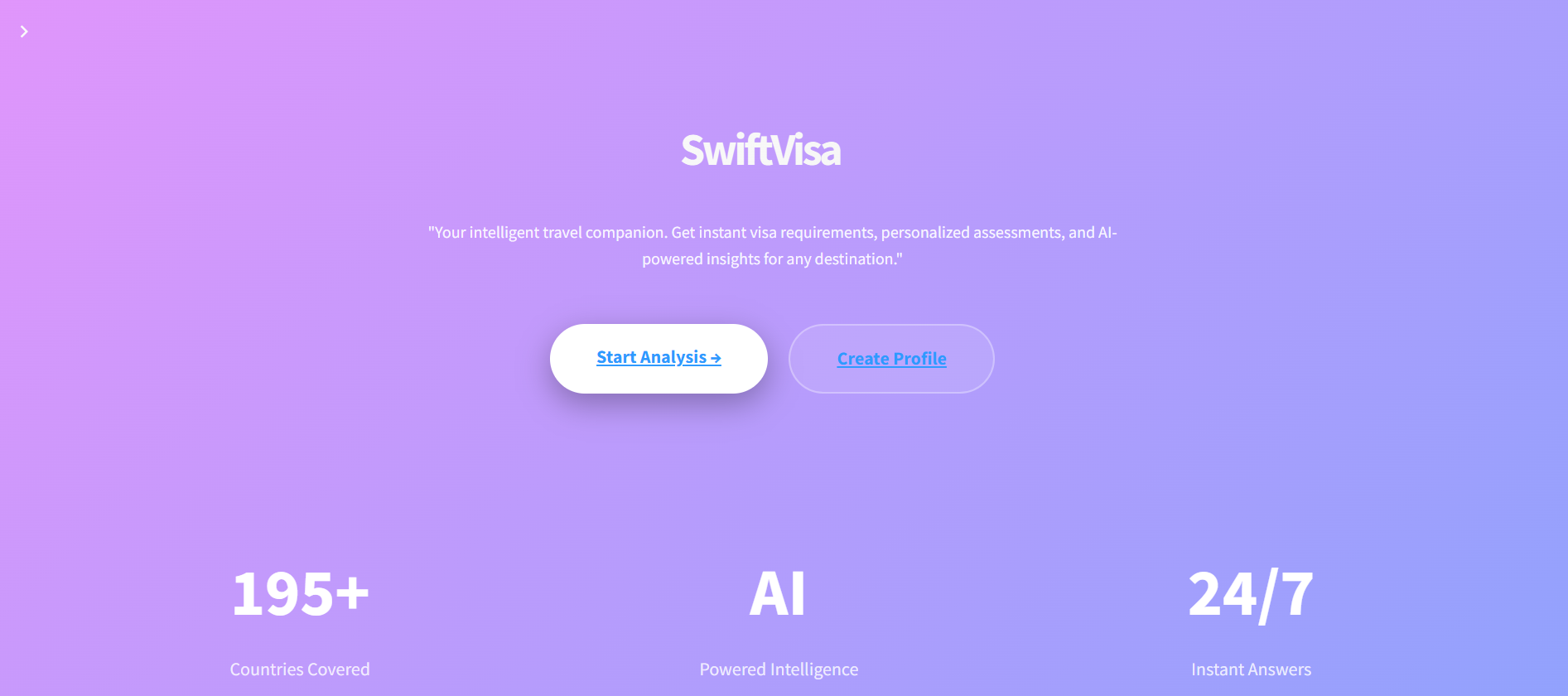
### **Table : User Interface Modules**

|  |  |
| --- | --- |
| **UI Module** | **Purpose** |
| Home Page | Introduction to SwiftVisa and navigation |
| User Profile Page | Collection of user-specific details |
| Visa Analyzer Page | Query submission and result display |
| Sidebar Navigation | Page navigation and system access |

## **8.4 Home Page Implementation**

The Home Page acts as the entry point of the SwiftVisa system. It introduces the purpose of the application and provides navigation options to other sections. Visual elements such as headings, icons, and interactive buttons enhance user engagement.

The home page also includes an interactive world map visualization that provides high-level visa information for different countries. This feature improves user awareness and encourages exploration.

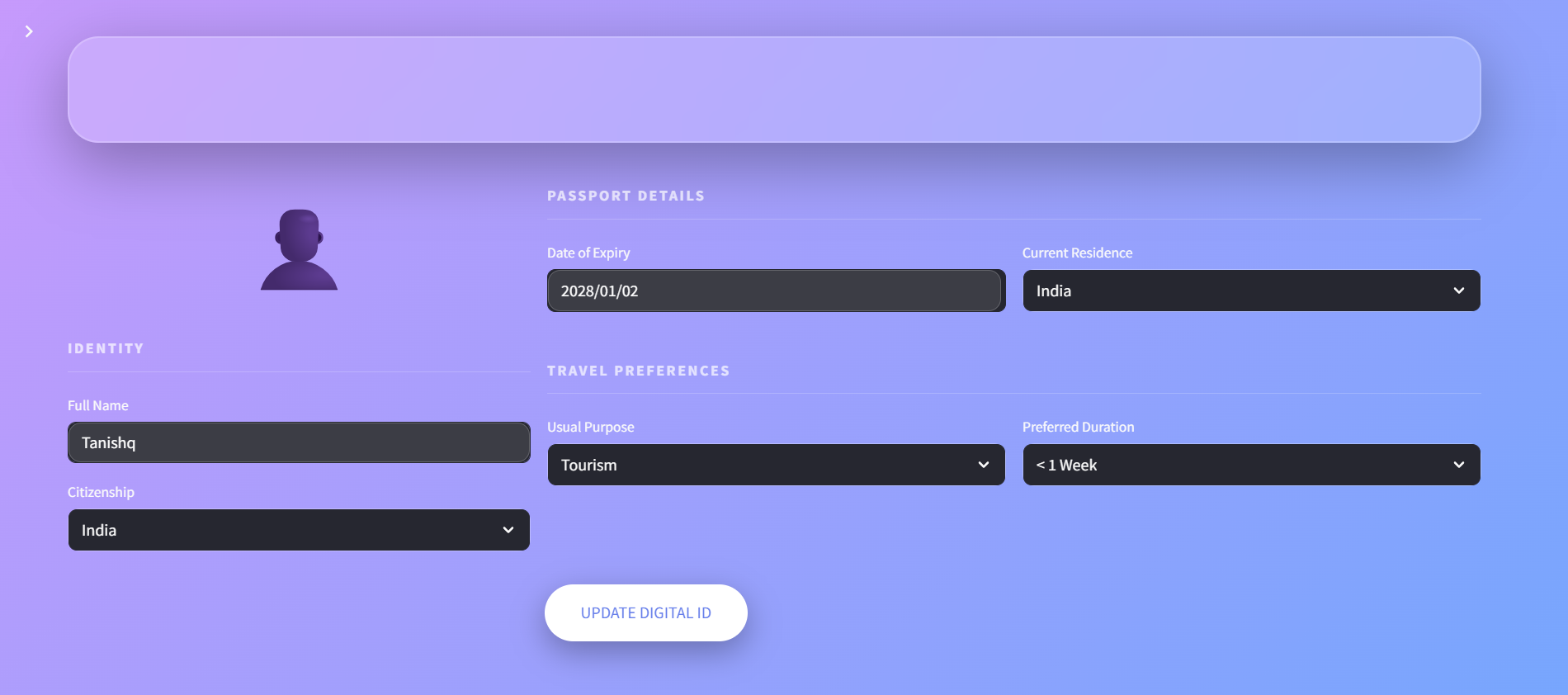


## **8.5 User Profile Page Implementation**

The User Profile page allows users to enter personal and travel-related details. This information is used to personalize visa explanations.

Key features of the profile page include:

* Structured input fields
* Validation of mandatory fields
* Temporary storage using session state
* Visual feedback on successful data entry



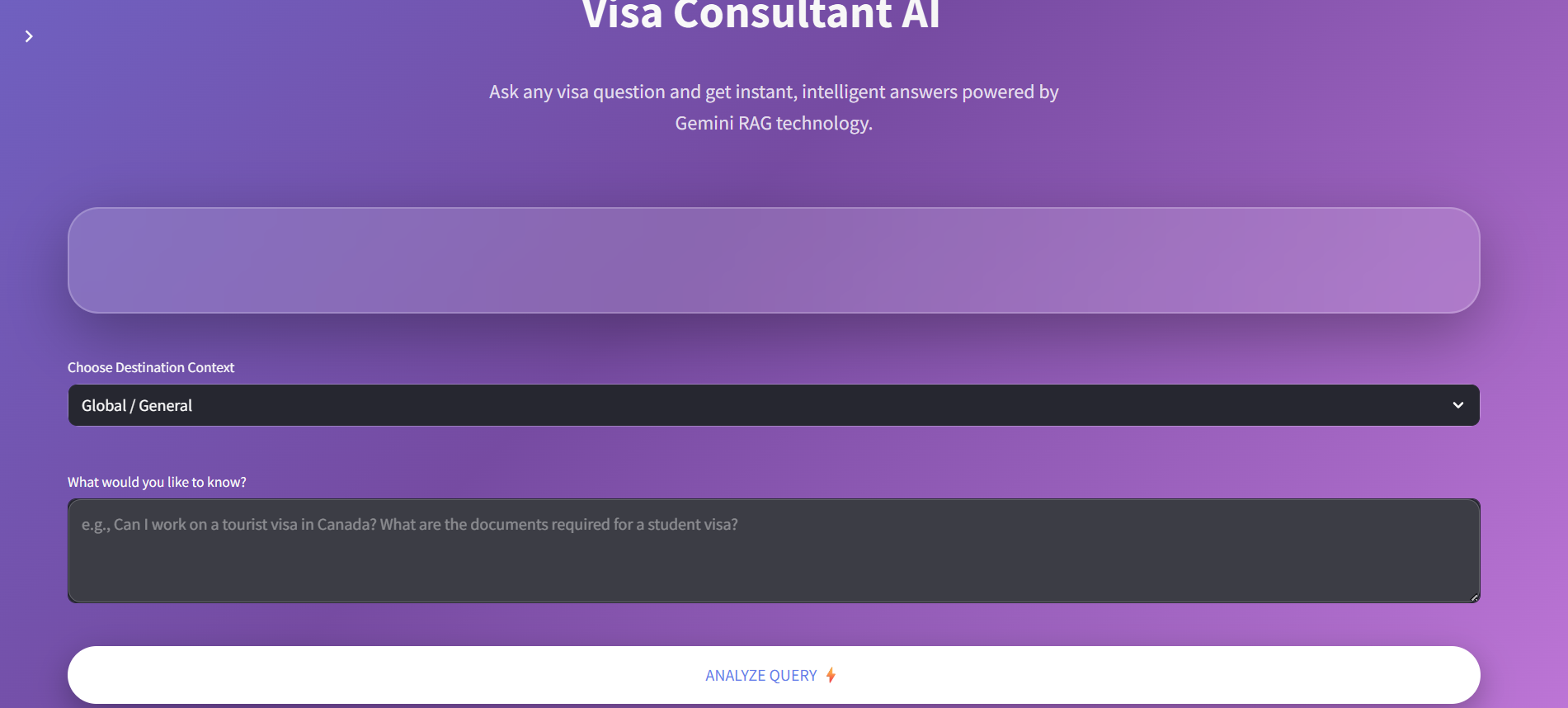
**8.6 Visa Analyzer Page**

The Visa Analyzer page is the core interaction module where users submit visa-related questions. This page integrates directly with the backend RAG pipeline.

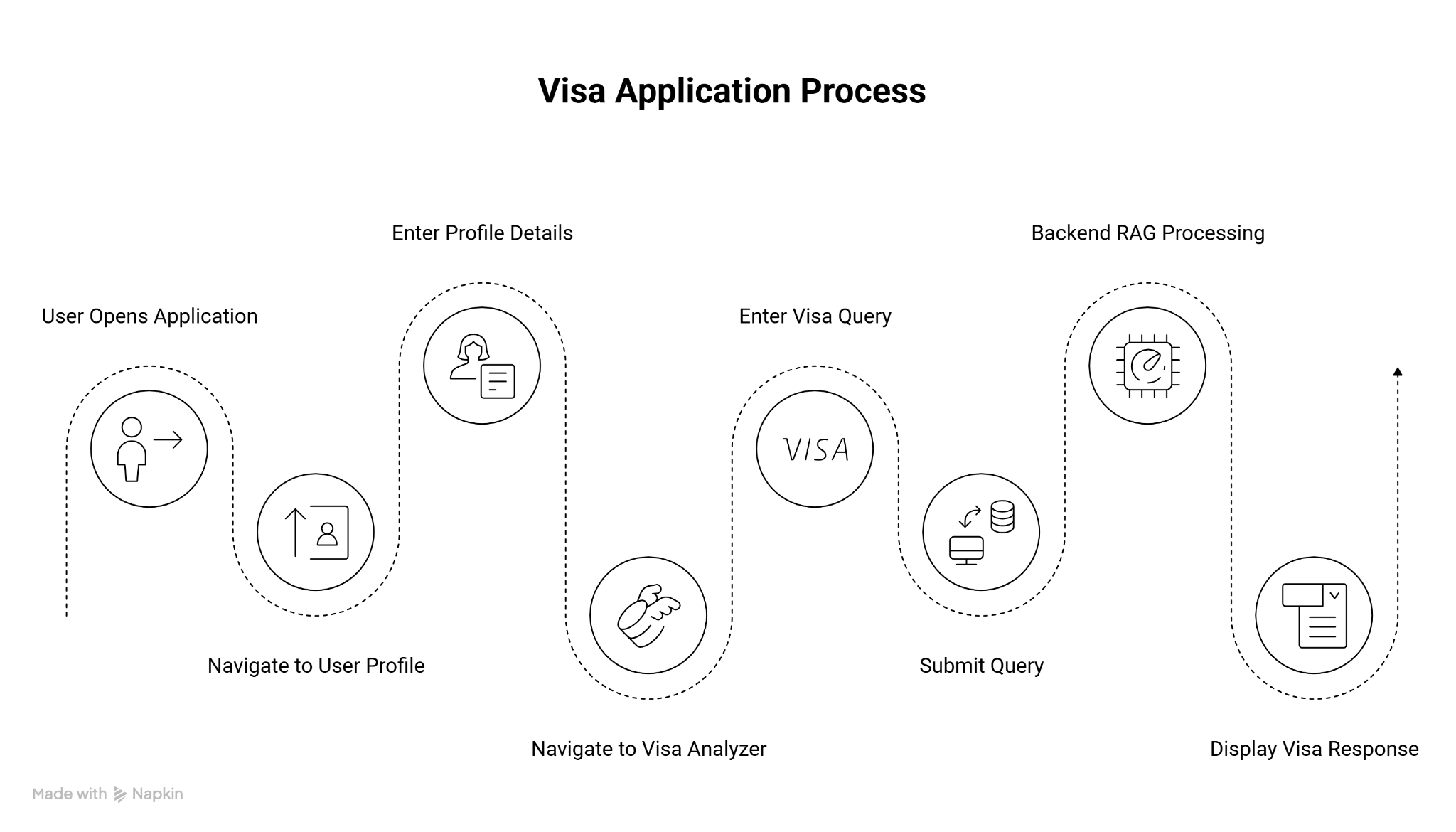
Users can:

* Select a country context
* Enter natural language queries
* Trigger analysis
* View generated responses

Real-time status indicators provide feedback during query processing, improving user experience.



### **Flowchart : User Interaction Flow**



## **8.6 Backend Integration with UI**

The Streamlit interface communicates with the backend through direct function calls. When the user submits a query, the system:

* Retrieves user profile data from session state
* Combines the query with selected country context
* Sends the request to the RAG pipeline
* Receives and displays the response

**Pseudocode**

1. Launch Streamlit application

2. Display navigation menu

3. If user selects Profile Page:

a. Display profile form

b. Store input in session state

4. If user selects Visa Analyzer:

a. Accept user query

b. Retrieve profile from session state

c. Call backend RAG function

d. Display returned response

5. Allow user to submit new queries

## 

## **8.7 Visual Design and User Experience**

The SwiftVisa interface follows a clean and modern design approach. A consistent color theme, readable typography, and responsive layout improve usability. Animations and visual feedback enhance user engagement without affecting system performance.

CHAPTER 9

**CONCLUSION**

This internship project successfully designed and implemented **SwiftVisa**, an intelligent visa screening assistant based on the Retrieval-Augmented Generation approach. The system addresses the limitations of traditional search methods and standalone language models by grounding responses in official visa policy documents. By combining semantic retrieval with controlled language model generation, SwiftVisa ensures accurate, reliable, and policy-aligned visa information.

The project demonstrated the effective use of document preprocessing, text chunking, embedding generation, vector database indexing, and prompt engineering to build a scalable and transparent information system. The integration of a user profile module enabled contextualized explanations without making eligibility decisions, while the Streamlit-based user interface provided an intuitive and interactive experience.

Overall, the SwiftVisa system highlights the practical application of modern AI techniques in policy-driven domains and demonstrates how Retrieval-Augmented Generation can be used to deliver trustworthy and user-friendly information services. The project fulfilled its objectives and provided valuable hands-on experience in building end-to-end AI-powered applications.