[GitHub Links](https://github.com/aishwaryanr/awesome-generative-ai-guide?tab=readme-ov-file" \l "book-list-of-free-genai-courses)

**Use Case 1: Customer Support Chatbot**

**Scenario**: A company wants to deploy an AI-powered customer support chatbot that can answer queries related to its products, pricing, troubleshooting, etc.

1. **User Query**:
   * The user submits a query, such as "How do I reset my password?"
2. **Prompt Catalog**:
   * The system checks the **Prompt Catalog** for an appropriate prompt template, such as "Guide for password reset."
   * This enriches the user query to something more structured that the LLM can understand, like "Provide a step-by-step guide for resetting a password for user accounts."
3. **RAG Pipeline**:
   * **Source Connector**: The system pulls data from multiple sources, such as the knowledge base (FAQs, product manuals), support ticket history, and forums.
   * **Extraction**: The RAG pipeline extracts relevant content, like password reset steps or troubleshooting tips from the knowledge base.
   * **Chunking**: If the reset instructions are lengthy, the system might chunk them into smaller, digestible pieces (e.g., "Step 1: Go to settings").
   * **Embeddings**: The extracted content is converted into embeddings (numerical vectors) for fast search and retrieval.
4. **Vector Database**:
   * The **Vector Database** stores the embeddings, enabling the system to retrieve content with semantic relevance (e.g., "How to reset your password").
   * When a query comes in, the system finds the nearest matching content in the vector database (password reset procedure) based on similarity to the user's query.
5. **LLM Gateway**:
   * The **LLM Gateway** takes the enriched prompt and uses an appropriate model (e.g., GPT-4) to generate a response, combining the user query and the retrieved context.
6. **Generated Response**:
   * The LLM generates a detailed and context-rich response: "To reset your password, follow these steps..."
7. **Feedback Loop**:
   * As users interact with the chatbot, the system may track interactions, ensuring the knowledge base and vector database are updated when new solutions or FAQs are created.

**Use Case 2: Document Summarization and Query**

**Scenario**: A legal firm wants to use AI to summarize and query documents (such as contracts, court rulings, etc.) quickly.

1. **User Query**:
   * The user submits a query like "Summarize this contract."
2. **Prompt Catalog**:
   * The system enriches the prompt to focus on the core objective: "Provide a high-level summary of the key terms in this contract."
3. **RAG Pipeline**:
   * **Source Connector**: The system connects to document repositories (e.g., contract storage systems, SaaS platforms).
   * **Extraction**: It extracts sections related to key terms, obligations, dates, and legal clauses.
   * **Chunking**: It breaks the long contract into smaller chunks (e.g., "Section 1: Terms and Conditions," "Section 2: Payment Schedule").
   * **Embeddings**: These chunks are converted into vectors so that the system can search for the most relevant sections in response to specific queries.
4. **Vector Database**:
   * The **Vector Database** stores all extracted chunks with embeddings, enabling the AI system to retrieve contract sections related to the query.
   * When the user requests a summary, it retrieves key clauses or points and provides them for context.
5. **LLM Gateway**:
   * The LLM Gateway uses the enriched prompt and relevant chunks to guide the LLM, which generates a concise and coherent summary of the contract.
6. **Generated Response**:
   * The LLM produces a summary: "This contract includes an agreement to pay $50,000 in installments over the next 12 months, with a clause specifying penalties for late payment."
7. **Monitoring and Updates**:
   * The document repository is continually monitored for changes (new contracts or updates), ensuring that the RAG pipeline has the latest information to process.

**Use Case 3: Healthcare AI Assistant**

**Scenario**: A healthcare provider wants to create an AI assistant that answers medical queries, such as symptoms, diagnoses, treatments, etc.

1. **User Query**:
   * The user submits a query: "What are the symptoms of diabetes?"
2. **Prompt Catalog**:
   * The system enriches the query: "Provide a list of common symptoms of Type 2 diabetes."
3. **RAG Pipeline**:
   * **Source Connector**: The AI connects to healthcare data sources like medical knowledge bases, research articles, and patient records.
   * **Extraction**: It extracts relevant information, such as symptoms from articles or medical guidelines.
   * **Chunking**: The information is chunked into smaller pieces—e.g., "Increased thirst," "Frequent urination."
   * **Embeddings**: This information is embedded into vectors.
4. **Vector Database**:
   * The embeddings are stored and indexed in the **Vector Database** to enable fast retrieval of similar information when a query is made.
5. **LLM Gateway**:
   * The system uses the enriched prompt and context (symptoms) to guide the LLM in generating a response.
6. **Generated Response**:
   * The LLM generates a detailed response: "Common symptoms of Type 2 diabetes include increased thirst, frequent urination, fatigue, and blurred vision."
7. **Feedback Loop**:
   * The system is continually updated with the latest medical guidelines and research, ensuring that the response remains current and accurate.

**Technical Insights into the System's Workflow**

* **Data Sources (Mostly Unstructured Data)**: The system needs to handle vast amounts of unstructured data, such as documents, images, and chat logs. These sources are often stored in systems like SaaS applications, file systems, or databases.
* **Real-time Data Updates**: The system watches for updates and changes in the connected data sources. For instance, if a new document or FAQ is added to a knowledge base, the system integrates it into the RAG pipeline to improve responses.
* **Scalability**: The use of vector embeddings enables the system to scale efficiently. Vector databases like FAISS or Pinecone allow for fast similarity searches across large data sets, which is crucial for handling queries in real time.
* **Embedding Algorithms**: Technologies like BERT, GPT, or custom-trained models can generate embeddings. These embeddings ensure that even if the user’s query uses different terminology, the system can still retrieve relevant context.