Knowledge Transfer - Using AI Assistant/Agent - Design Plan

1. Overview

Purpose

The purpose of this app is to assist users in:

- **Document-based Q&A**: Users can query the app about uploaded documents.
- **Podcast Generation**: Generate an audio summary or deep-dive podcast based on document content.
- **Knowledge Transfer Support**: Provide easy access to summarized, organized, and structured information for team sessions.

Target Audience

- Project teams for knowledge transfer sessions.
- Educators/trainers delivering workshops.
- Students or professionals needing efficient document review.

2. Features

Core Features

1. **Document Upload**
    - Users can upload files in various formats: TXT, PDF, PPTX, PPT, DOCX, CSV.
    - File size limit: 200MB.
2. **Document Q&A**
    - Allow users to ask questions related to the uploaded document content.
    - Natural language understanding using an LLM-based retrieval system.
3. **Podcast Generation**
    - Generate audio content (deep-dive or summary) from documents.
    - Use **text-to-speech (TTS)** to create audio output.
    - Support customization like podcast duration or section selection.
4. **Voice Query**
    - Users can interact with the app using a **voice-based interface**.
    - Speech-to-text for query input.

5. **Multi-Agent System** (for next level functionality)
    - Add agents for specific tasks:
        - **Summarization Agent**: Provides concise summaries.
        - **FAQ Generation Agent**: Automatically generates FAQs based on document content.
        - **Study Guide Agent**: Creates study notes or flashcards.
        - **Timeline Agent**: Extracts timelines for historical or process-based documents.
6. **Chat History & Notes**
    - Store user queries, responses, and podcast summaries as **notes**.
    - Downloadable notes in PDF/CSV format.
7. **Customizable Interface**
    - Allow users to customize Q&A formats, podcast style, and notes layout.

3. System Architecture

3.1 High-Level Design

1. **Frontend** (Streamlit):
    - **User Interface (UI)**: Upload file, input queries, view responses, control podcast generation.
    - **Audio Playback**: Use `streamlit_audio` or equivalent to play generated podcasts.
    - **Interactive Notes**: Streamlit forms to add, edit, and save notes.
2. **Backend Services**
    - **File Parsing**: Process and extract text from uploaded documents.
    - **RAG Pipeline** (Retrieval-Augmented Generation):
        - Index document content using embeddings (e.g., OpenAI, Sentence Transformers).
        - Perform similarity-based retrieval for Q&A.
    - **Text-to-Speech (TTS)**: Generate audio summaries using tools like:
        - Google Text-to-Speech (gTTS)
        - Azure Speech Service
        - OpenAI's TTS APIs.
    - **Speech-to-Text (STT)**: Convert voice input into queries using:
        - OpenAI Whisper
        - Google Speech Recognition API.
3. **Database**
    - Store uploaded files, user queries, responses, generated podcasts, and notes.
    - **Technology**: SQLite (local) or PostgreSQL (cloud).
4. **Model Integration**
    - Use **OpenAI GPT-4/Gemini 2.0** or any preferred LLM for generating responses.

- For embeddings, integrate **FAISS/Chroma** or vector databases like Pinecone.

4. User Flow

Step 1: File Upload

- User uploads a document (TXT, PDF, etc.).
- The document is parsed, and embeddings are created for retrieval.

Step 2: Document Q&A

- User types or speaks a question.
- The app retrieves relevant sections using similarity search and generates a response.

Step 3: Podcast Generation

- User clicks **"Generate Podcast"**.
- Select podcast style (summary, deep-dive).
- TTS API generates audio output for playback.

Step 4: Notes Management

- User can add manual notes or download chat history.

5. UI/UX Design

Main UI Components

1. **File Upload Box**
   - Drag-and-drop functionality.
   - Display uploaded file name, size, and status.
2. **Q&A Chat Box**
   - Input field for user query.
   - Real-time voice-to-text option with a microphone button.
   - Response area to display generated answers.
3. **Podcast Generator**

    - Button to trigger podcast creation.
    - Audio playback controls.
4. **Notes Panel**
    - Add, view, and download notes.

Streamlit Libraries/Components to Use

- **st.file_uploader**: For file uploads.
- **st.text_input & st.text_area**: For user queries and notes.
- **st.button**: For podcast generation.
- **st.audio**: For audio playback.
- **st.expander**: To organize UI sections.

6. Technology Stack

Component	Technology
Frontend	Streamlit, Streamlit Audio, HTML/CSS
Backend	Python (FastAPI for API calls)
LLM Integration	OpenAI GPT-4 or Gemini 2.0(for production)/GroqCloud(for development purposes), HuggingFace Transformers
Embeddings	OpenAI Embeddings, VertexAIEmbeddings, HuggingFaceEmbeddings
Text-to-Speech	gTTS, Azure TTS, Google TTS
Speech-to-Text	OpenAI Whisper, Google STT
Database	SQLite/PostgreSQL
File Parsing	PyPDF2, python-docx, csv

7. Deployment Plan

1. **Local Development**:
    - Use Streamlit to test functionality locally.
    - Integrate the backend APIs for file handling and LLM calls.
2. **Cloud Deployment**:
    - Deploy the app on **Streamlit Cloud** or **GCP** (Google Cloud Run).
    - Use a cloud database (PostgreSQL).
3. **API Hosting**:
    - Host backend APIs for LLM, embedding, and TTS generation on **FastAPI**.

8. Future Enhancements

- **Multi-Language Support**: Allow Q&A and podcasts in multiple languages.
- **Integration with Cloud Drives**: Fetch documents directly from Google Drive or OneDrive.
- **Collaboration Features**: Multiple users can collaborate on the same session.
- **Advanced Analytics**: Provide insights like word counts, key phrases, and document trends.
- **Document Summarization Pipelines**: Add multi-level summaries (bullet points, paragraph, headline).