HACKATHON PROJECT PHASES TEMPLATE FOR THE STUDY MATE PROJECT

Hackathon Project Phases Template

Project Title:

StudyMate: An AI-Powered PDF-Based Q&A System for Students

Team Name:

Team Squad

Team Members:

- JakkiReddy Sushma
- Vadavalli Bhavani
- Tadigadapa Revathi

Phase 1: Project Planning and Conceptualization

Objective:

To design and develop an AI-powered educational tool that allows users to upload PDFs and interact with them through a chatbot interface. The chatbot should answer user questions based on the content of the PDF using advanced Natural Language Processing (NLP) techniques.

Key Points:

- Use AI to make studying from documents easier.
- Enable real-time Q&A from PDF content.
- Integrate Hugging Face and Google Gemini models.
- Simple and interactive web-based interface using Streamlit.

- Context-aware response generation using semantic search

1.Problem Statement:

Students, researchers, and professionals often deal with large, complex PDF documents. Reading through entire documents to find relevant information is time-consuming and inefficient. There is a lack of tools that allow users to interactively query a PDF and get instant answers in natural language.

2.Proposed Solution:

- Create **StudyMate**, an intelligent chatbot platform where users can:
  Upload PDF files.
  Ask questions in natural language.
  Receive context-based or general answers depending on the PDF content.

The system uses:

- **PyMuPDF** for reading PDFs.
- **Sentence Transformers** and **FAISS** for chunking and semantic search.
- **Flan-T5** (Hugging Face) for contextual responses.
- **Google Gemini API** for general question answering.

3.Target Users:

- **Students:** preparing for exams or studying notes
- **Researchers:** working with lengthy technical papers.
- **Professionals:** dealing with manuals, legal docs, or reports.
- **Educators:** who want to simplify content delivery.

4.Expected Outcome:

- An AI-powered chatbot that simplifies document navigation.
- Accurate, fast answers from uploaded PDFs.
- User-friendly chat interface for academic and professional use.
- A scalable tool that can evolve into a personalized AI tutor.

Phase 2: Requirement Analysis

Objective:

Define the technical and functional requirements for the **StudyMate** AI-powered PDF Q&A application.

Key Points:

1. **Technical Requirements:**

   Programming Language: Python

   Frontend: Streamlit Web Framework

   PDF Processing: PyMuPDF (fitz)

   Embedding Model: SentenceTransformer (all-MiniLM-L6-v2)

   Vector Indexing: FAISS (Facebook AI Similarity Search)

   Answer Generation:

   Hugging Face Flan-T5 (google/flan-t5-base)

   Google Gemini Flash API (gemini-1.5-flash)

2. **Functional Requirements:**

   Allow users to upload PDFs and extract clean text from them.
   Enable semantic chunking of large documents for better processing.
   Perform vector embedding and similarity search for context retrieval.
   Answer user questions using either contextual models or general models.
   Present results through a clean chat-style interface using Streamlit.
   Maintain session-based chat history for ongoing Q&A experience.

3. **Constraints & Challenges:**

   - Managing performance for large PDFs and long user queries.
   - API rate limits from Google Gemini or Hugging Face.
   - Ensuring relevant and accurate responses, even with vague questions.
   - Optimizing embedding and indexing speed for real-time usage.
   - Delivering a smooth and responsive user experience in the Streamlit UI.

Phase 3: Project Design

Objective:

Develop the system architecture, user flow, and UI/UX design for the **StudyMate** chatbot application.

Key Points:

1. **System Architecture:**
   User uploads a PDF file through the Streamlit UI.

   The PDF is processed and split into chunks using PyMuPDF and text chunking logic.

   Each chunk is embedded using a SentenceTransformer model.

   A FAISS index is built for similarity search.

   The user enters a natural language question.

   The app searches for the most relevant chunks using the FAISS index.

   If context is found, the Hugging Face Flan-T5 model generates a response.

   If no context is found, the Gemini Flash API provides a general answer.

   The frontend displays the Q&A chat history in real time.

2. **User Flow:**

   Step 1: User uploads a PDF document via the Streamlit interface.
   Step 2: The app extracts and processes the content into manageable chunks.
   Step 3: User types a question (e.g., *"What are the key findings in this report?"*)
   Step 4: The app retrieves relevant chunks and generates an answer.
   Step 5: The answer is displayed in the chat interface along with previous interactions**.**

3. **UI/UX Considerations:**

   Clean, chat-style interface for intuitive user interaction.
   Loading indicators for background processing (e.g., "Reading PDF...")

Text input box with a “You:” prompt for simplicity.
Session-based chat memory for a natural conversation flow.
Minimalist design for academic users with support for mobile and desktop layouts.
Footer credits and branding ("Powered by Hugging Face & Gemini · Built for CognitiveX")

Phase 4: Project Planning

Objective:

Break down the development of StudyMate into sprints using Agile methodology to ensure efficient and collaborative project execution.

Sprint Planning with Priorities
Sprint 1 – Setup & UI (Day 1)
🔴 **High Priority** – Set up development environment, install Python libraries (PyMuPDF, Streamlit, Hugging Face, FAISS).
🔴 **High Priority** – Integrate Gemini and Hugging Face APIs with token keys.
🟡 **Medium Priority** – Build initial Streamlit layout with file uploader and input box.

Sprint 2 – Core Functionality (Day 2)
🔴 **High Priority** – Implement PDF processing, chunking, and embedding.
🔴 **High Priority** – Add semantic search and conditional answer generation logic.

Sprint 3 – Final Touches & Deployment (Day 2)
🟡 **Medium Priority** – Add session-based memory, refine chat display and error handling.
🟢 **Low Priority** – Prepare final demo video and deploy project on local/hosted server

---

Phase 5: Project Development

Objective:

Implement the core features of the StudyMate application with a focus on PDF-based question answering using AI models

Key Points:

1. **Technology Stack Used:**

   Frontend: Streamlit

   Backend: Hugging Face Transformers, Google Gemini Flash API (Optional)

   Embedding & Semantic Search: Sentence Transformers, FAISS

   PDF Parsing: PyMuPDF (fitz)

   Programming Language: Python

Development Process:

API Integration & Setup:
Integrated Google Gemini Flash and Hugging Face API with secure key management.
PDF Upload & Parsing:
Users upload academic PDFs; content is extracted using PyMuPDF.
Text Chunking & Embedding:
The extracted text is divided into manageable chunks and converted into embeddings using Sentence Transformers.
Similarity Search with FAISS:
When a user asks a question, the app performs semantic search using FAISS to find relevant chunks.
Answer Generation:
The top-ranked chunks are passed to a generative model (Flan-T5 or Gemini Flash) to formulate human-like responses.

Challenges & Fixes:
Challenge: Slow response time for long PDFs
Fix: Implemented chunking + embedding offline; reused FAISS index for fast search.
Challenge: Irrelevant or generic answers from LLM
Fix: Added context window tuning and prompt optimization to narrow the question scope.
Challenge: Handling memory in Streamlit sessions
Fix: Used st.session_state to store chat history and embeddings during a session.

Phase-6: Functional & Performance Testing

Objective:

Validate the core functionalities, performance speed, and UI responsiveness of the StudyMate application.

Functional Testing:

Query from PDF (e.g., *"What is OSI Model?"*) responded correctly.
English queries interpreted and answered accurately.

Performance Testing:

PDF embedding and search slightly slow (~8 seconds); needs optimization.

Bug Fixes:

Fixed inaccurate answer mapping by improving FAISS indexing.

UI Validation:

UI is responsive on desktop but breaks on mobile view.

Deployment Testing:

Successfully deployed on Streamlit; publicly accessible.