Hackathon Project Phases Template

Project Title: StudyMate: AI-Powered Academic Assistant

Team Name: Team SAR

Team Members: S.Revanth,G.Sai Nehal, Y.Akshay

Phase-1: Brainstorming & Ideation

Objective:
To develop "StudyMate," an AI-powered academic assistant that allows students to interact with their study materials in a conversational Q&A format, alongside providing document summarization and a glossary of key terms.

Key Points:

- **Problem Statement:** Students often struggle with passively reading large and dense academic documents like textbooks, lecture notes, and research papers. Manually searching for specific information is time-consuming and inefficient.
- **Proposed Solution:** An AI-powered application that enables users to upload PDF documents and ask questions in natural language. The application will provide direct, context-aware answers sourced from the uploaded content. It will also offer features for document summarization and the generation of a glossary of important terms.
- **Target Users:**
    - Students who need to quickly find information in their study materials.
    - Researchers who need to review and analyze multiple research papers efficiently.
    - Anyone looking to better understand complex academic or technical documents.
-
- **Expected Outcome:** A functional AI-powered academic assistant that enables conversational Q&A with PDF documents, provides concise summaries, and generates a helpful glossary, all based on the user's uploaded content.

Phase-2: Requirement Analysis

Objective:
To define the technical and functional requirements for the StudyMate application.

Key Points:

- **Technical Requirements:**
    - **Programming Language:** Python

    - **Frontend:** Streamlit
    - **LLM for Answer Generation:** IBM Watsonx's Mixtral-8x7B-Instruct
    - **Text Extraction:** PyMuPDF
    - **Embeddings:** HuggingFace SentenceTransformers
    - **Vector Store:** FAISS (Facebook AI Similarity Search)
    - **Summarization & Glossary:** Integrated LLM functionalities.
-
- **Functional Requirements:**
    - Allow users to upload one or more PDF documents.
    - Enable users to ask questions in a conversational format.
    - Provide accurate and contextually relevant answers based on the uploaded documents.
    - Generate a summary of the uploaded document(s).
    - Create a glossary of key terms found within the text.
    - Display answers and other generated content in a clear and user-friendly interface.
-
- **Constraints & Challenges:**
    - Ensuring accurate text extraction from various PDF layouts and formats.
    - Managing the computational resources for generating embeddings and performing similarity searches, especially with large documents.
    - Handling potential API rate limits and optimizing calls to the IBM Watsonx model.
    - Maintaining a smooth and responsive user experience with Streamlit, even with complex backend processes.
-

Phase-3: Project Design

Objective:
To develop the architecture and user flow of the StudyMate application.

Key Points:

- **System Architecture:**
    - **Frontend (Streamlit):** The user uploads PDF(s) and inputs questions through the web interface.
    - **Backend (Python):**
        - **Text Extraction:** PyMuPDF extracts text from the uploaded PDFs.
        - **Text Chunking:** The extracted text is divided into smaller, manageable chunks.
        - **Embedding Generation:** HuggingFace SentenceTransformers convert text chunks into vector embeddings.
        - **Vector Storage:** FAISS creates an index of these embeddings for efficient semantic search.
    -

    - **Information Retrieval:** When a user asks a question, their query is converted into an embedding. FAISS then retrieves the most relevant text chunks based on semantic similarity.
    - **Answer & Content Generation:** The retrieved text chunks are passed as context to the IBM Watsonx Mixtral-8x7B-Instruct model, which generates the final answer, summary, or glossary.
    - **Display:** The generated content is sent back to the Streamlit frontend to be displayed to the user.
-
- **User Flow:**
    - **Step 1:** The user opens the web application and is prompted to upload one or more PDF documents.
    - **Step 2:** After the documents are processed, the user can type a question into the input field.
    - **Step 3:** The user can also navigate to tabs or sections for "Document Insights" to view the summary and glossary.
    - **Step 4:** The application processes the query and displays the answer, summary, or glossary in a designated area of the interface.
-
- **UI/UX Considerations:**
    - A clean, intuitive interface that makes document upload and question asking straightforward.
    - Clear separation of functionalities, possibly using tabs for "Q&A," "Summarization," and "Glossary."
    - A loading indicator to provide feedback to the user while backend processes are running.
    - The ability to easily clear the current session and upload new documents.
-

Phase-4: Project Planning (Agile Methodologies)

Objective:
To break down the development tasks into manageable sprints for efficient completion.

Sprint	Task	Priority	Duration	Deadline	Assigned To	Dependencies	Expected Outcome
Sprint 1	Environment Setup & PDF Processing	🔴 High	4 hours (Day 1)	Mid-Day 1	Shanawaz	Python, Streamlit, PyMuPDF	PDF text extraction and chunking functional.

Sprint 1	Basic Frontend UI Development	🟡 Medium	2 hours (Day 1)	End of Day 1	Mohammad	Streamlit	Basic UI with file uploader and text input.
Sprint 2	Embedding & Vector Store Integration	🔴 High	4 hours (Day 2)	Mid-Day 2	Anwar	FAISS, HuggingFace	Semantic search functionality is working.
Sprint 2	LLM Integration for Q&A	🔴 High	2 hours (Day 2)	Mid-Day 2	Shanawaz	IBM Watsonx API	Answer generation from retrieved context.
Sprint 3	Summarization & Glossary Feature	🟡 Medium	2 hours (Day 2)	End of Day 2	Mohammad	LLM integration	Summaries and glossaries are generated.
Sprint 3	Testing, UI/UX Refinements & Deployment	🟢 Low	2 hours (Day 2)	End of Day 2	Entire Team	Working prototype	A polished, demo-ready project.

Sprint Planning with Priorities:

- **Sprint 1 – Setup & Core Backend (Day 1):**
  - (🔴 High Priority) Set up the development environment and install all necessary dependencies.
  - (🔴 High Priority) Implement PDF text extraction and preprocessing using PyMuPDF.
  - (🟡 Medium Priority) Build the initial Streamlit UI for file uploading and user input.
-
- **Sprint 2 – AI Integration & Search (Day 2):**
  - (🔴 High Priority) Integrate HuggingFace for embeddings and FAISS for semantic search.
  - (🔴 High Priority) Connect to the IBM Watsonx API for question-answering.
-
- **Sprint 3 – Feature Enhancement & Finalization (Day 2):**
  - (🟡 Medium Priority) Implement summarization and glossary generation features.

    - (🟢 Low Priority) Conduct thorough testing, refine the user interface, and prepare for the final presentation and deployment.
-

Phase-5: Project Development

Objective:
To implement the core features of the StudyMate application.

Key Points:

- **Technology Stack Used:**
    - **Frontend:** Streamlit
    - **Backend & Orchestration:** Python
    - **LLM:** IBM Watsonx (Mixtral-8x7B-Instruct)
    - **Text Extraction:** PyMuPDF
    - **Embeddings & Search:** HuggingFace SentenceTransformers, FAISS
-
- **Development Process:**
    - Develop a robust text extraction pipeline with PyMuPDF.
    - Implement a text chunking strategy to create meaningful segments for embedding.
    - Set up the FAISS vector store to index and search the document embeddings.
    - Integrate the IBM Watsonx API, ensuring that retrieved text chunks are passed as context in the prompt.
    - Develop the additional functionalities for summarization and glossary generation.
-
- **Challenges & Fixes:**
    - **Challenge:** Inaccurate text extraction from PDFs with complex layouts (e.g., multi-column documents, tables).
    - **Fix:** Implement more advanced parsing logic and potentially use OCR as a fallback for scanned documents.
    - **Challenge:** The context provided to the LLM may not be sufficient or could be noisy.
    - **Fix:** Experiment with different chunk sizes and overlapping strategies to improve the quality of retrieved context.
-

Phase-6: Functional & Performance Testing

Objective:
To ensure that the StudyMate application functions correctly and performs efficiently.

Test Case ID	Category	Test Scenario	Expected Outcome	Status	Tester
TC-001	Functional Testing	Upload a multi-page PDF and ask a specific question from the later pages.	A correct and contextually relevant answer should be generated.	✅ Passed	Shanawaz
TC-002	Functional Testing	Request a summary of an uploaded research paper.	A concise and accurate summary should be provided.	✅ Passed	Anwar
TC-003	Functional Testing	Generate a glossary for a textbook chapter.	A list of key terms and their definitions should be displayed.	✅ Passed	Mohammad
TC-004	Performance Testing	Measure the response time from asking a question to receiving an answer for a 50-page document.	The response time should be within an acceptable range (e.g., under 15 seconds).	⚠ Needs Optimization	Anwar
TC-005	UI/UX Testing	Test the application on different screen sizes to ensure responsiveness.	The UI should adapt well to both desktop and mobile views.	⚠ Needs Optimization	Mohammad
TC-006	Deployment Testing	Deploy the app using Streamlit Sharing or another hosting service.	The application should be accessible and fully functional via a public URL.	🚀 Deployed	Shanawaz

---

Final Submission

- **GitHub/Code Repository Link:** https://github.com/Revanth1902/SAR_CognitiveX_GenAI_Hackathon
- **Demo Video (3-5 Minutes):** https://drive.google.com/file/d/1MS2botCwmV7PXYSWntj8HY0iNh5ipcog/view?usp=sharing