Hackathon Project Phases

Project Title:

JobSwift: Accelerating Careers with AI-Powered Applications using Gemini API

Team Name:

LYNX

Team Members:

- K.Adharsh
- M.Rakesh
- M.Rohini
- A.Sai Meghamsh
- M.Raj Mehathah

Phase-1: Brainstorming & Ideation

Objective:

- Identify the problem statement.
- Define the purpose and impact of the project.

Key Points:

1. **Problem Statement:** (What problem are you solving?)

   Job seekers often struggle to create effective, impactful resumes that highlight their skills and experience to potential employers. The resume creation process can be time-consuming, daunting, and require specialized writing skills, leading to missed opportunities. Many existing resume templates are generic and don't leverage modern AI capabilities to optimize content for specific job roles or industries.

2. **Proposed Solution:** (Briefly explain your idea)

   JobSwift is an AI-powered platform that simplifies and accelerates resume creation through an interactive chat-based interface. Users engage in a conversational Q&A session, providing their career information. The platform then utilizes Google's Gemini model to generate a personalized, optimized resume tailored to their profile, leveraging AI to enhance content and structure for better impact.

3. **Target Users:** (Who will benefit from this project?)
    - Job seekers at all career levels (entry-level to experienced professionals).
    - Individuals looking to quickly create or update their resumes.
    - Those who find resume writing challenging or lack confidence in their resume writing skills.
    - Users who want to leverage AI to create a more effective and modern resume.

4. **Expected Outcome:** (What will the project achieve?)
    - A functional, user-friendly web platform (JobSwift) accessible via a web browser.
    - A chat-based resume builder interface using Streamlit.
    - AI-powered resume generation capabilities using Google's Gemini model in a FastAPI backend.
    - A streamlined and efficient resume creation process for users.
    - Improved resume quality and effectiveness for job seekers, potentially leading to better job application outcomes.
    - A foundation for future expansion with features like cover letter generation and job application tracking.

Phase-2: Requirement Analysis

Objective:

- Define technical and functional requirements.

Key Points:

1. **Technical Requirements:** (Languages, frameworks, tools)
    - **Frontend:** Streamlit (Python-based framework for UI).
    - **Backend:** FastAPI (Python-based framework for API and backend logic).
    - **AI Model Integration:** Google Gemini API through `google-generativeai` Python library.
    - **API Communication:** `requests` library for frontend to backend API calls.
    - **Development Environment:** VS Code, Replit, or Google Colab (for development).
    - **Deployment :** Cloud platform like Replit Deployment

2. **Functional Requirements:** (Features the project must have)
    - **Core Feature:** Chat-based resume building interface.
    - **AI Resume Generation:** Backend processing user input with Gemini to generate resume content.
    - **Real-time Preview:** Display evolving resume in the frontend.
    - **Download Resume:** Option to download the generated resume (initially plain text, potentially PDF/DOC in later iterations).
    - **Basic Backend API Endpoints:**
    - /chat_resume endpoint to receive chat data and return generated resume content.
    - /test_resume (for initial connection testing).
    - **User Interface (UI):** Intuitive, responsive chat interface in Streamlit.

3. **Constraints & Challenges:** (Any limitations or risks)
    - **AI Model Accuracy & Quality:** The quality of the generated resume depends on the Gemini model's performance and prompt engineering. Ensuring generated resumes are relevant, professional, and error-free is crucial.
    - **API Rate Limits & Costs:** Google Gemini API usage might have rate limits or costs associated. Need to be mindful of API usage, especially during development and potential scaling.
    - **Data Privacy & Security:** Handling user-provided resume information securely is important. (Though for this simplified version without database persistence, data is primarily in-memory and transient).
    - **User Experience (UX) Design:** Designing a chat flow that is both engaging and effectively captures necessary resume data will require iterative design and testing.
    - **Frontend-Backend Integration:** Ensuring seamless and reliable communication between the Streamlit frontend and FastAPI backend.

- **Error Handling:** Implementing robust error handling in both frontend and backend to provide informative feedback to users and developers.

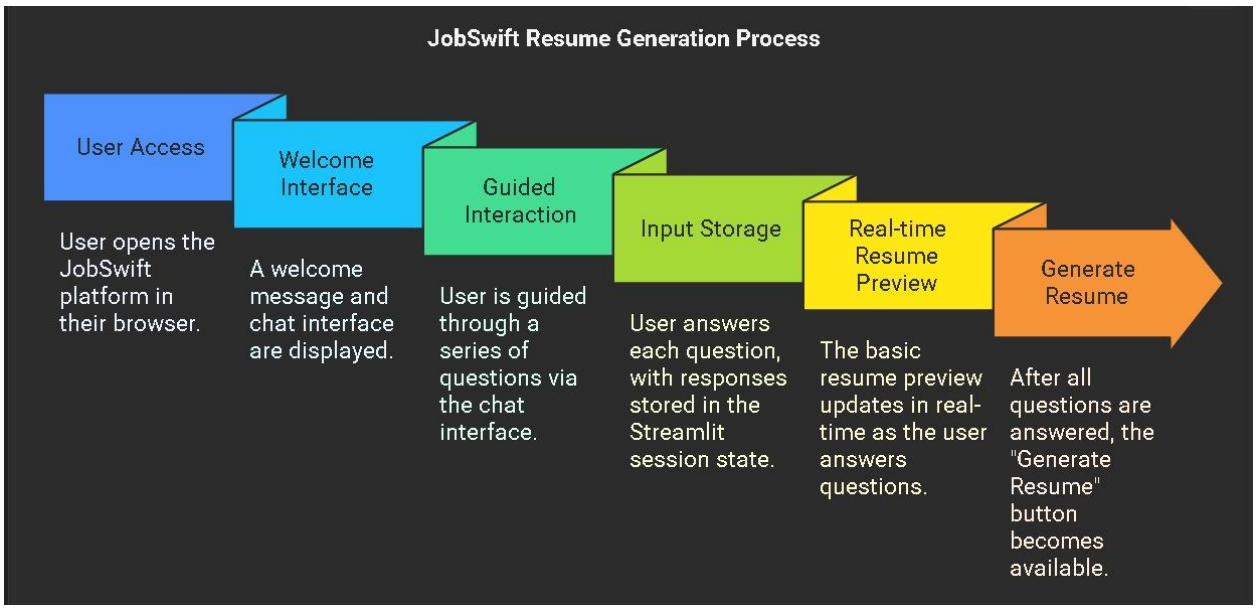
Phase-3: Project Design

Objective:

- Create the architecture and user flow.

Key Points:

1. **System Architecture Diagram:** (Simple sketch or flowchart)



2. **User Flow:** (How a user will interact with the project)
    - User Opens JobSwift Platform (Streamlit Frontend in Browser).
    - Welcome message and chat interface is displayed.
    - User is guided through a series of questions via the chat interface (Streamlit `st.chat_input`, `st.chat_message`).
    - User answers each question, input is stored in Frontend (Streamlit session state).
    - Basic Resume Preview updates in real-time as user answers questions (Streamlit UI update).
    - After answering all questions, "Generate Resume" button becomes available.
    - User clicks "Generate Resume" button.
    - Frontend sends collected user data (conversation/resume data) to FastAPI Backend's `/chat_resume` endpoint (HTTP POST request using `requests`).
    - Backend receives data, constructs prompt, calls Google PaLM API (text-bison-001).

- Gemini model generates resume content, Backend receives response.
- Backend sends the generated resume content back to the Frontend (HTTP response with JSON payload).
- Frontend receives resume content, displays it in the "Resume Preview" area (formatted display in Streamlit).
- Frontend provides "Download" options (initially plain text download, later PDF/DOC).
- User can download the resume.
- Session ends (data is not persistently saved in this simplified version).

3. **UI/UX Considerations:** (If applicable, wireframe or basic layout)
    - **Chat Interface Focus:** Clean, intuitive chat interface as the primary interaction method.
    - **Real-time Preview:** Prominent display area for the evolving resume preview, clearly showing the generated content as users provide information.
    - **Clear Question Prompts:** Questions should be easy to understand and guide the user effectively through resume sections.
    - **Progress Indication:** Visual cues (e.g., progress bar, question numbering) to show users how far they are in the resume building process.
    - **Download Options:** Clearly visible download buttons with format options.
    - **Responsive Design:** The UI should be responsive and work well on different screen sizes (desktop, tablets).
    - **Visual Appeal:** Use Streamlit's theming and styling options to create a professional and visually appealing interface. (To be refined in later sprints).

Phase-4: Project Planning (Agile Methodologies)

Objective:

Break down the tasks using Agile methodologies.

Sprint	Task (Minimized)	Priority	Duration	Deadline	Assigned To	Expected Outcome (Minimized)
Sprint 1	**Basic Setup & Connection:** Project setup, UI & Backend stubs, test connection	High	8 Hours	End of Day 1		Basic setup and confirmed frontend-backend communication

Sprint 2	**Core Resume Generation:** Chat UI questions, Gemini integration & AI resume output	High	4 hours	End of Day 1		Functional chat UI and AI-powered resume generation displayed
Sprint 3	**Refine & Basic Polish:** Prompt refinement, UI improvements, basic testing	High	2 Hours	End of Day 2		Improved resume quality, basic UI polish, and tested core functionality
Sprint 3	Final Presentation & Deployment	High	1 hour (Day 2)	End of Day 2	Entire Team	Demo-ready project

Key Points:

- Sprint 1 (Combined Setup & Basic Connection): Combines the initial setup tasks (project setup, libraries) with creating the *basic* UI and backend, and testing the frontend-backend connection. This sprint focuses on getting the foundational elements in place in a shorter timeframe.
- Sprint 2 (Core Functionality Sprint - More Intensive): This is the most intensive sprint. It combines:
    - Expanding the Streamlit UI to the full chat-based Q&A flow.
    - Integrating the Google Gemini API in the backend.
    - Implementing the core logic for prompt construction and calling the Gemini model.
    - Ensuring the AI-generated resume is successfully passed from the backend to the frontend and displayed. This sprint focuses on building the *core* feature of the platform - the AI-powered resume generation through chat. It's given a longer duration (4 days) to accommodate the complexity.
- Sprint 3 (Refinement & Polish Sprint): Merges the refinement and testing tasks:
    - Improving the quality of AI-generated resumes by refining Gemini prompts.
    - Enhancing the UI/UX for better user experience.
    - Implementing the basic download functionality.
    - Crucially, it also includes the testing, bug fixing, polishing, and documentation from the original

Phase-5: Project Development

Objective:

- Code the project and integrate components.

Key Points:

1. **Technology Stack Used:** (List of programming languages, APIs, etc.)
    - **Programming Languages:** Python (for both Frontend and Backend).
    - **Frontend Framework:** Streamlit.
    - **Backend Framework:** FastAPI.
    - **AI Model API:** Google Gemini API (text-bison-001) accessed through `google-generativeai` Python library.
    - **API Communication:** HTTP requests using `requests` library in Frontend.
    - **Development Environment:** VS Code/Replit (or Colab for initial prototyping).

2. **Development Process:** (Steps followed for coding)
    - **Iterative Development:** Follow Agile Sprint plan, building and testing incrementally sprint by sprint.
    - **Frontend-First UI Design (for initial sprints):** Start by building the basic Streamlit UI shell and chat flow.
    - **Backend API Endpoint Development:** Develop FastAPI endpoints incrementally, starting with placeholders and then adding AI logic.
    - **Integration Testing (Sprint by Sprint):** Test integration between Frontend and Backend after core features of each sprint are implemented.
    - **Code Versioning (Recommended):** Use Git and a platform like GitHub (or Replit's built-in version control) to track code changes.
    - **Modular Coding:** Organize code into logical modules (e.g., separate files for frontend, backend, models, utilities - even if database is skipped for now, keep structure).

3. **Challenges & Fixes:** (Mention any obstacles faced and how they were solved)
    - **(Potential Challenge): Streamlit UI Responsiveness with Complex Chat Flows:** May need to optimize Streamlit code or UI structure for smooth updates as chat history grows. *(Fix strategy: Profile Streamlit performance, optimize code, consider UI patterns for large chat histories if needed.)*
    - **(Potential Challenge): PaLM API Prompt Engineering:** Crafting effective prompts to get high-quality resume content from Gemini model requires experimentation and refinement. *(Fix strategy: Iterative prompt design, testing different prompt structures, analyzing* Gemini *outputs, refining prompts based on results. Explore prompt engineering techniques.)*
    - **(Potential Challenge): Frontend-Backend Communication Issues:** Debugging API request/response flow, handling data serialization/deserialization, error

handling between frontend and backend. *(Fix strategy: Use browser developer tools to inspect network requests, implement logging in backend and frontend, use consistent data formats (JSON), implement clear error messages.)*

- **(Potential Challenge): Gemini API Errors and Rate Limits:** Handling potential errors from the Gemini API (network errors, API quota limits, etc.). *(Fix strategy: Implement robust error handling in backend API calls to* Gemini*, implement retry logic with exponential backoff, monitor API usage, consider caching if appropriate, handle rate limit errors gracefully and inform user.)*

Phase-6: Functional & Performance Testing

Objective:

- Ensure the project works as expected.

Key Points:

1. **Test Cases Executed:** (List the scenarios tested)
2. **Bug Fixes & Improvements:** (Mention fixes made)
3. **Final Validation:** (Does the project meet the initial requirements?)
4. **Deployment (if applicable):** (Hosting details or final demo link)

Test Case ID	Category	Test Scenario	Expected Outcome	Status
TC-JS-001	Functional Testing	Start chat, answer all Q&A prompts with valid data	Chat conversation flows smoothly, "Generate Resume" button appears after last Q	✅ Passed
TC-JS-002	Functional Testing	Provide empty input for a question in the chat interface	System prompts user to provide a valid answer, does not proceed without input	✅ Passed
TC-JS-003	Functional Testing	Click "Generate Resume" button after completing chat	AI-generated resume content is displayed in the preview section	✅ Passed
TC-JS-004	Functional Testing	Review generated resume content for relevance to provided inputs	Resume content is relevant to user's answers and logically structured	✅ Passed

TC-JS-005	Functional Testing	Refresh the browser during a chat session	Chat history and previously entered data are preserved using session state	✅ Passed

---

Final Submission

1. **Project Report Based on the templates**
2. **Demo Video (3-5 Minutes)**
3. **GitHub/Code Repository Link**
4. **Presentation**
