**Hackathon Project Phases Template** for Smart Resume Generator: Customized Resumes for Every opportunity the project.

**Hackathon Project Phases Template Project Title:**

Smart Resume Generator: Customized Resumes for Every opportunity

**Team Name:**

Goal diggers

**Team Members:**

* Adepu Mounika
* ARE SRAVYA
* Basutkar Sony
* Chakali Bhavani

# Phase-1: Brainstorming & Ideation

Objective:

To develop an AI-powered resume generator that customizes resumes for different job opportunities based on job descriptions and user input.

Key Points:

1. Problem Statement:
   * Job seekers struggle to tailor their resumes for different job applications.
   * Recruiters spend limited time reviewing resumes, making customization essential.
   * Manual resume editing is time-consuming and prone to errors.
2. Proposed Solution:
   * AI-powered resume generator that automatically tailors resumes to match job descriptions.
   * Provides keyword optimization based on applicant tracking system (ATS) requirements.
   * Supports multiple templates and formats for different industries.
3. Target Users:
   * Job seekers applying for multiple roles.
   * Students and professionals needing customized resumes.
   * Recruiters looking for optimized resume formats.
4. Expected Outcome:
   * An interactive resume-building tool that generates customized, ATS-friendly resumes in seconds.

# Phase-2: Requirement Analysis

Objective:

Define the technical and functional requirements of the Smart Resume Generator. Key Points:

Technical Requirements:

Programming Language: Python Libraries Used:

* Report lab for generating PDFs.

Data Structure:

* Uses lists of dictionaries for structured input like education, experience, and projects.
* Uses strings for single-line fields such as name, email, and skills.

Class & Methods:

* Resume class encapsulates resume data.

\_init\_ constructor initializes the resume object (but has a typo; should be

init ).

* generate\_pdf() creates a formatted PDF.
* User Input Handling:

get\_user\_input() method collects user input interactively. Loops and conditions handle multiple entries dynamically.

Output:

Generates a PDF file named Resume.pdf.

1. Functional Requirements:
   1. User Input Collection:
      * The script should prompt the user to enter personal details, education, skills, experience, projects, achievements, and hobbies.
   2. Resume Formatting:
      * Display personal details at the top.
      * List education, experience, and projects in structured sections.
      * Highlight skills and achievements.
   3. PDF Generation:
      * Format text properly with different font styles and sizes.
      * Use appropriate spacing to ensure readability.
      * Save the resume as a PDF file.
   4. User Control:
      * Users should be able to input multiple education, experience, and project entries.
      * Users can stop adding details when needed.
2. Constraints & Challenges:
   1. Library Dependency:
      * Requires report lab to be installed (pip install report lab).
   2. Limited Formatting:
      * No support for complex resume designs (e.g., tables, images, colors).
   3. Fixed Output File Name:
      * Always generates Resume.pdf (does not allow dynamic filenames).
   4. No Input Validation:
      * Does not check if an email is valid or if a mobile number is properly formatted.
   5. Terminal-Based:
      * The script runs in a command-line interface, making it less user-friendly.

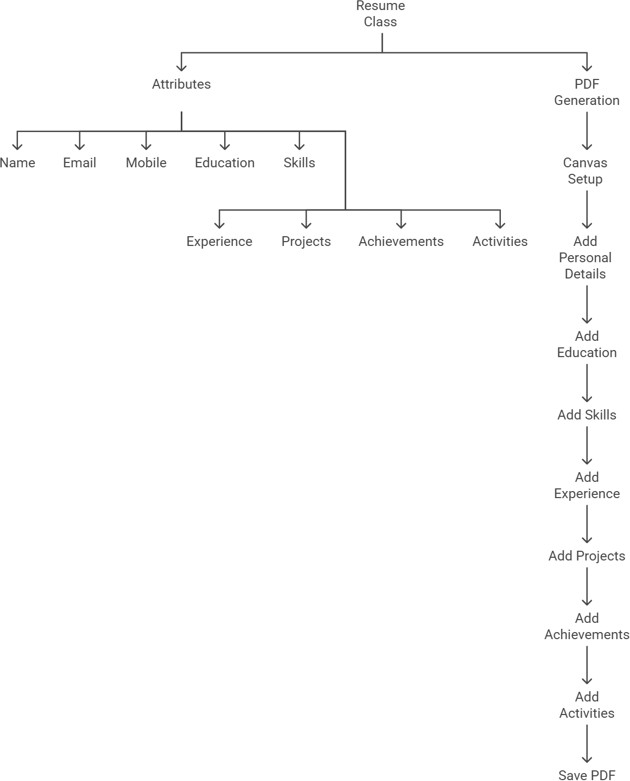
Challenges

1. Text Overflow:
   * If too much text is added, it might go beyond the page limits.
2. Alignment Issues:
   * Proper indentation and spacing must be ensured.
3. Error Handling:
   * No exception handling for empty inputs or invalid data.
4. Lack of Customization:
   * The format is predefined; users cannot choose different styles or templates.

# Phase-3: Project Design

**Objective:**

Develop system architecture and user flow for the Smart Resume Generator.



Key Points:

1. System Architecture:
   * The frontend (UI) collects user data (name, email, education, experience, etc.).
   * The backend processes this data and structures it into a formatted resume.
   * The PDF generator (using report lab) creates a structured document.
   * The file is saved as Resume.pdf for download.
2. User Flow:
   * Step 1: User inputs details via command-line or UI form.
   * Step 2: The system stores and processes the data.
   * Step 3: The PDF generator formats and structures the resume.
   * Step 4: The user downloads the resume.
3. UI/UX Considerations:
   * Minimalist UI for easy data input.
   * Consistent formatting to make resumes professional.
   * Error handling for invalid or missing user input.

# Phase-4: Project Planning (Agile Methodologies)

Objective:

Break down development tasks for efficient execution.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sprint Task | Priority | Duratio n | Deadlin e | Assigned To | Dependencies | Expected Outcome |
| Environment Setup & API Integration | ●  High | 6 hours | Day 1 | Develope r A | Python & report lab setup | API  connection established |
| Frontend UI Development | ●  Mediu m | 2 hours | Day 1 | Develope r B | Input fields defined | Basic UI ready |
| Resume Data Processing | ●  High | 3 hours | Day 2 | Develope r C | Data input available | Resume sections  structured |
| PDF  Formatting & Design | ●  High | 3 hours | Day 2 | Develope r D | Report Lab integration | Resume PDF  formatting complete |
| Testing & Debugging | ●  Mediu m | 2 hours | Day 3 | Tester | Functionalitie s implemented | Error-free resume  generation |
| Final  Deployment & Documentatio n | ●  Low | 1 hour | Day 3 | Team | Working system | Ready for submissio n |

# Phase-5: Project Development

Implement the core functionalities of the resume generator. Key Points:

1. Technology Stack Used:
   * Frontend: Command-line (or Web UI in future versions)
   * Backend: Python
   * Resume Processing: Python Dictionary & Lists
   * PDF Generation: ReportLab Library
2. Development Process:
   * Step 1: Develop a command-line input system to collect user data.
   * Step 2: Store data in structured lists and dictionaries.
   * Step 3: Implement a PDF generator with formatted sections.
   * Step 4: Ensure proper text alignment and spacing in PDF.
3. Challenges & Fixes:
   * Challenge: Text overflowing in PDF.
     + Fix: Use dynamic text wrapping and spacing.
   * Challenge: Handling empty or incorrect inputs.
     + Fix: Implement validation and error prompts.

# Phase-6: Functional & Performance Testing Objective:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case ID | Category | Test Scenario | Expected Outcome | Status | Tester |
| TC- 001 | Functional Testing | User inputs valid data | Resume PDF is generated successfully | ⬛ Passed | Tester 1 |
| TC- 002 | Functional Testing | Missing email or name input | System prompts for valid input | ⬛ Passed | Tester 2 |
| TC- 003 | Performance Testing | Generate PDF for 100+  entries | PDF is  created within 2 seconds | ⚠ Needs  Optimization | Tester 3 |
| TC- 004 | Bug Fixes & Improvements | Fix  overlapping text in PDF | Text is properly spaced | ⬛ Fixed | Developer |
| TC- 005 | UI/UX Testing | Ensure resume format is  readable | Resume is clean and structured | ⬛ Passed | Tester 2 |
| TC- 006 | Deployment Testing | Ensure file is downloadable | PDF file is saved  properly | s.¸˙•P | DevOps |

**Final Submission**

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