# SDLC Case Study Worksheet

**Project Title:** Resume Analyzer (AI-based)

**Team Name:** CerebroSparks

**Team Members and Roles:**

* Wajeeha Batool **(Project Manager)**
* Usaid Ahmed **(Frontend Developer)**
* Anabiyah Ahmed **(Backend Developer)**
* Shahana Jamal **(AI Specialist)**
* Fareeha Jawed **(QA Engineer)**

## 1. Requirements Phase

### Functional Requirements:

1. **Resume parsing and data extraction:** The system must be able to accept resume files in common formats (e.g., PDF, DOCX) and accurately extract key information, such as skills, education, and work experience.
2. **Keyword Matching:** Identifies keywords within resumes and compares them to the requirements of a job description.
3. **Skill matching and Scoring:** The core algorithm must compare the skills extracted from a resume against the keywords identified in the job description to calculate a relevance score.
4. **Feedback generation:** The system must provide actionable feedback to the user based on the comparison. For a job seeker, this could highlight missing keywords or formatting issues. For a company, it could show a ranking of candidates.
5. **User interface for input and output:** The tool must have a user-friendly interface where users can upload a resume and paste a job description. It must also display the analysis results, score, and feedback in a clear format.

Non-Functional Requirements:

1. **Performance:** The system must process a resume and job description and deliver a score within an acceptable timeframe (e.g., within 10–20 seconds).
2. **Security:** The system must protect sensitive user data, such as personal information from resumes and job application history. This includes implementing data encryption and access controls.

## 2. Design Phase

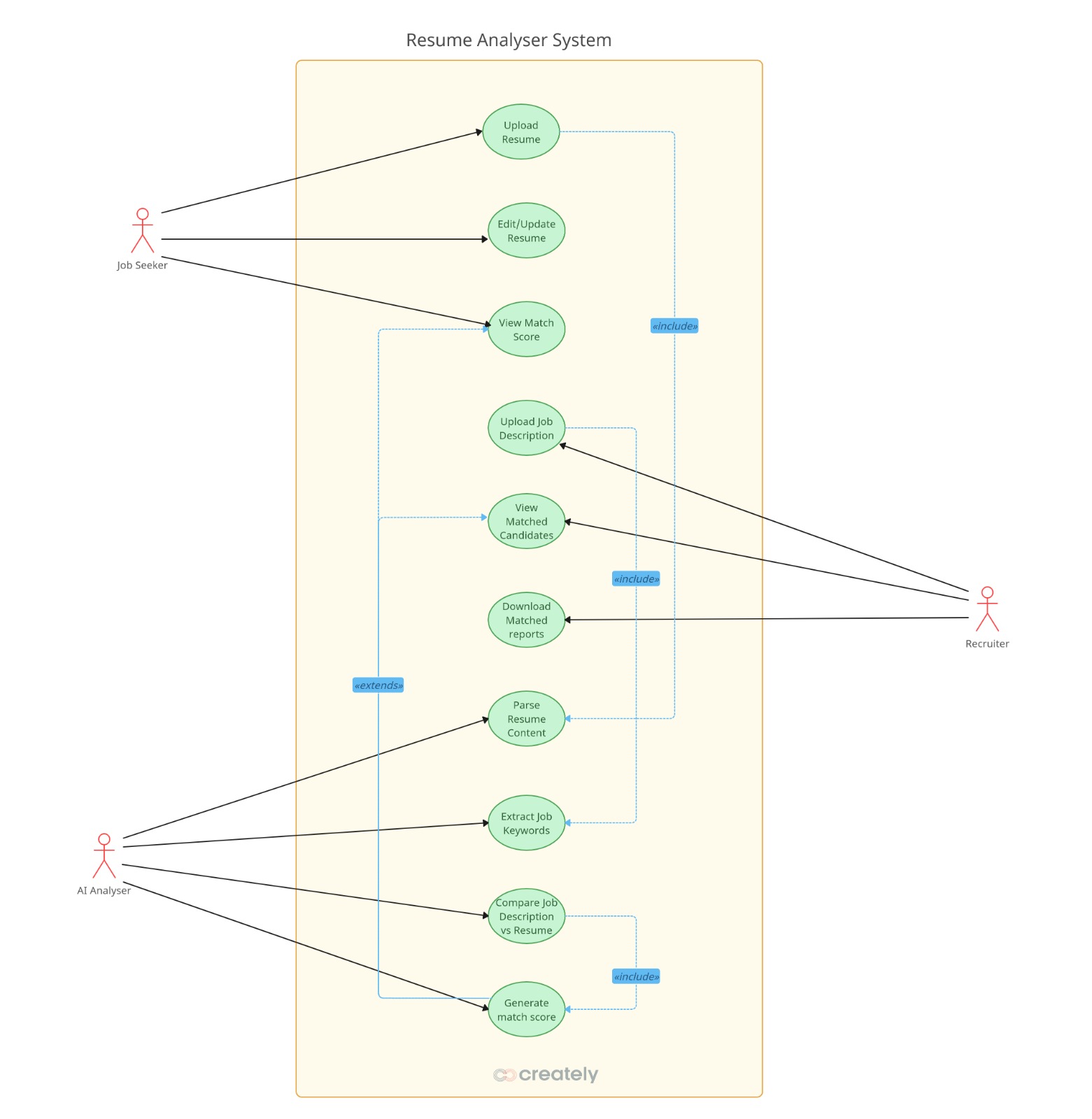
### - Work Breakdown Structure (WBS)

**Resume Analyzer Project**

* **1.1 Project Planning**
* 1.1.1 Project Scope Definition
* 1.1.2 Technology Stack Selection
* 1.1.3 Identify and confirm stakeholders
* **1.2 Requirements Gathering**
* 1.2.1 Requirement Specifications Document
* 1.2.2 Document Functional and Non-Functional requirements
* **1.3 Backend Development**
  + 1.3.1 Resume Parsing Module
  + 1.3.2 Skill Matching Algorithm (Model training)
  + 1.3.3 Database Designing and Development
  + 1.3.4 API Endpoints Development
* **1.4 Frontend Development**
* 1.4.1 User Interface Design (UI)
* 1.4.2 Resume and Job Upload Interface
* 1.4.3 Results Display Interface
* **1.5 Integration**
* 1.5.1 Backend-Frontend Integration
* 1.5.2 AI Model API Integration
* **1.6 Testing**
* 1.6.1 Unit Testing
* 1.6.2 Integration Testing
* 1.6.3 AI Performance & Output Testing
* 1.6.4 User Acceptance Testing (UAT)
* **1.7 Deployment and Maintenance**
* 1.7.1 Production Environment Setup
* 1.7.2 Application Deployment
* 1.7.3 Post-launch Monitoring

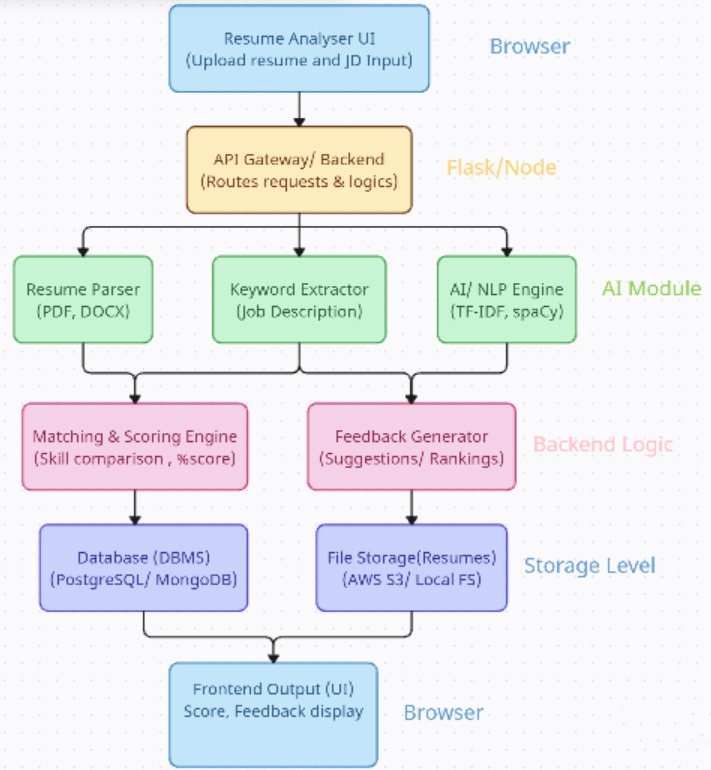
### - UML (Use Case Diagram)

**Tool**: [Creately | Visual Collaboration & Diagramming Platform](https://creately.com/?aff=h2tmedia)



## 3. Backend Design

**Tool**: [Creately | Visual Collaboration & Diagramming Platform](https://creately.com/?aff=h2tmedia)



## 4. Development Phase

Pseudo-Code of key function: Skills Matching

## **FUNCTION name:** Skill\_Matching

## **INPUT(s):** Resume text, Job description text

## **PROCESS:**

## Extract skills from both inputs.

## Compare skills from resume with job description keywords.

## Count matched keywords.

## Compute percentage of match.

## (matched\_keywords / total\_keywords) \* 100

## **OUTPUT**: Display matched skills and match score.

## **END FUNCTION**

## Testing Phase

|  |  |
| --- | --- |
| **Test Case ID** | **Description** |
| TC01 | Check if the system correctly calculates match percentage between resume and job description. |
| TC01 | Verify system detects missing skills in the resume. |
| TC01 | Ensure the system rejects empty or invalid file uploads. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case ID** | **Input(s)** | **Expected Output** | **Result (Pass/Fail)** |
| TC01 | **Resume:** Python, Java, SQL, and Data Analysis.  **Job Description:** Python, SQL, Java, Data Analysis. | A high match score (e.g., > 90%) | **Pass** |
| TC02 | **Resume:** Flutter and mobile app development.  **Job Description:** flutter, mobile app development. | A medium match score is displayed. | **Pass** |
| TC03 | **Resume:** File with no technical skills, only soft skills like **Communication, Teamwork, and Leadership**.  **Job Description:** Required Skills: Python, SQL, Java. | A very low (or zero) match score is displayed | **Pass** |

## 6. Reflection

1. Which SDLC phase was the most challenging? Why?  
   **Backend integration & data extraction** were the toughest. Parsing real-world resumes is messy, PDFs, DOCX, different templates, and mixed languages all behave differently. Even after converting files to text, separating **skills** from **tools** and **duties** required careful rules and a skills taxonomy to avoid false matches (e.g., matching **SQL** inside **Sequel of a project name**). On top of that, keeping the processing time within the **10–20s** budget while doing file upload, parsing, normalization, scoring, and feedback in one flow meant I had to design asynchronous jobs and clear status updates back to the UI. In short: **data quality and performance** made this phase the most complex.
2. Which SDLC model (Waterfall, Agile) best fits this project? Why?  
   **Agile** fits better. The core of a resume analyzer is an evolving **matching/feedback loop**: we ship a basic extractor, test on real resumes, then iterate on edge cases (acronyms, synonyms, soft vs. hard skills). That learning cycle benefits from short sprints, quick demos, and frequent updates to the skills taxonomy and scoring weights. Also, stakeholders (students/recruiters) give qualitative feedback that can change priorities quickly—perfect for agile ceremonies and incremental releases.
3. How you determine functional and non-functional requirements?  
   I started from the **core user journey** in the doc, upload resume + paste JD → get **score + feedback** and broke it into backend capabilities: file handling, parsing, skill extraction, matching, and feedback output. That became the **functional** list. For **non-functional** parts, I used constraints implied by the student/recruiter experience in the doc, **reasonable latency (10–20s)**, **security for PII**, reliability, and clear error messages, so the system feels safe and responsive. I also aligned them with the WBS items (backend modules, API endpoints, AI performance testing) so they’re testable.

## 7. Attachments

Attached design documents include:

- UML diagrams (use case and class diagrams):

* All UML.docx
* Backend\_ArchitectureDesign.docx