RESUME PARSER AND SKILL EXTRACTOR SYSTEM

CASE STUDY

Submitted To:

Cognizant Technology Solutions

Submitted By:

Asif Khan

Vanshika Sharma

Rambabu Yalgala

Sukanya Duddela

Mohd Fahad

Project Duration:

1st July, 2025 – 31st July, 2025

Track:

AWS AIA

Contents

Problem Statement:

In today’s competitive job market, organizations and academic institutions receive a large volume of resumes regularly. Manually reviewing and extracting relevant information like candidate details, technical and soft skills, and experience from these resumes is a time-consuming and error-prone task. Recruiters often face difficulties in identifying the right talent quickly due to inconsistent resume formats, unstructured data, and limited time. There is a strong need for an intelligent, scalable, and automated solution that can parse resumes, extract key information, and generate structured candidate profiles to accelerate the hiring or evaluation process.

Objective:

The objective of this project is to build an end-to-end **Resume Parser and Skill Extractor** that automates the resume handling workflow. Using **AWS Lambda, Python, and S3**, the system aims to:

- Automate the upload and processing of resumes (PDF format)
- Extract structured information (name, email, phone, skills, experience)
- Identify and classify both technical and non-technical skills.
- Generate clean and searchable candidate profiles in JSON/CSV formats
- Provide an optional web-based dashboard for report generation and analytics

Ultimately, the goal is to reduce manual workload, improve skill matching accuracy, and enable faster decision-making.

Scope of the Project:

This project covers the full lifecycle of resume processing with the following functional scope:

1. **Resume Upload & Cloud Integration:**
    - Design a user-friendly HTML form for uploading PDF resumes
    - Integrate with AWS S3 to store uploaded files securely
    - Validate file types and enforce naming conventions
2. **Text Extraction & Normalization:**
    - Trigger AWS Lambda automatically on new resume upload

- Use pdfminer.six or PyMuPDF to extract and clean raw text
- Handle corrupted or unreadable files with error logging

3. **Skill Extraction & Classification:**
    - Identify skills using keyword dictionaries or regular expressions
    - Optionally enhance accuracy using spaCy or AWS Comprehend for NLP tagging
    - Categorize skills into domains (Programming, Tools, Soft Skills, etc.)
    - Assign relevance or frequency scores
4. **Candidate Profile Generation:**
    - Combine all extracted data into a structured profile
    - Store profiles as JSON or CSV in S3
    - Assign unique candidate IDs and optionally anonymize data
5. **Admin Dashboard & Reports (Optional):**
    - View parsed resumes and generated profiles
    - Search/filter candidates by skill, experience, or ID
    - Export reports in CSV format
    - Track uploads, parsing errors, and success rates
    - Optional login/authentication for admin access
6. **Technologies & Methodologies:**
    - Backend: Python 3.x
    - AWS: S3, Lambda, DynamoDB, AWS Comprehend
    - Libraries: Boto3, pdfminer.six,
    - Dashboard: Flask (optional)
    - Project Methodology: Agile (2-week sprint cycles)

Requirement Analysis

1. Functional Requirements:

The system comprises the following modules, each responsible for a specific function:

- **Resume Upload and S3 Integration**

   Create and configure an **AWS S3** bucket to store uploaded resumes.

   Build a **Flask-based frontend** to upload resumes (PDF format only).

   Implement file type validation and enforce naming standards.

   Provide confirmation or error status after file submission.

- **PDF Parsing and Text Extraction**

   Configure an **AWS Lambda** function to trigger on S3 upload.

   Extract text using libraries like **PyMuPDF** or **pdfminer.six.**

   Clean and normalize the raw text data.

   Log and report errors for corrupt or unreadable files.

- **Skill Extraction and Classification**

   Use keyword-based, regex, and LLM/NLP-based methods to detect skills.

   Classify skills into categories: Programming, Tools and Platforms, Soft Skills

   Calculate the frequency or relevance of skills per candidate.

   Output structured data in JSON format.

- **Candidate Profile Generation**

   Combine parsed data into a complete profile.

   Each profile includes: Name, Email, Phone Number, Identified Skills, Summarized Experience (via LLM)

   Assign a unique Candidate ID.

   Save profiles in JSON or CSV format to DynamoDB.

- **Admin Dashboard and Reporting**

   Build a dashboard to view uploaded resumes and parsed profiles.

   Enable filtering/searching candidates by skills or experience.

   Allow download of filtered or complete reports in CSV.

   Track overall system metrics: Total uploads, Errors, Success rates

2. Technical Requirements

- **Programming Language**

  Python is used for all backend scripts, Lambda functions, and dashboard logic.

- **AWS Services**

  · **Amazon S3** – For uploading and storing resumes and profile outputs.

  · **AWS Lambda** – For serverless, automatic processing of resumes on upload.

  · **DynamoDB** – To store structured profile metadata.

  · **AWS Comprehend** – For entity recognition and NLP-based tagging.

  · **AWS CloudWatch** – To monitor Lambda logs and system performance.

- **Libraries and APIs**

  · **pdfminer.six**, **PyMuPDF** – For extracting text from PDFs.

  · **Boto3** – AWS SDK for integrating S3, Lambda, and DynamoDB.

  · **Flask** – Used to build the upload UI and the optional analytics dashboard.

  · **spaCy / AWS Comprehend** – For advanced NLP tagging (optional).

  · **Google Gemini API** – For LLM-powered experience summarization.

- **Data Formats**

  Input: .pdf

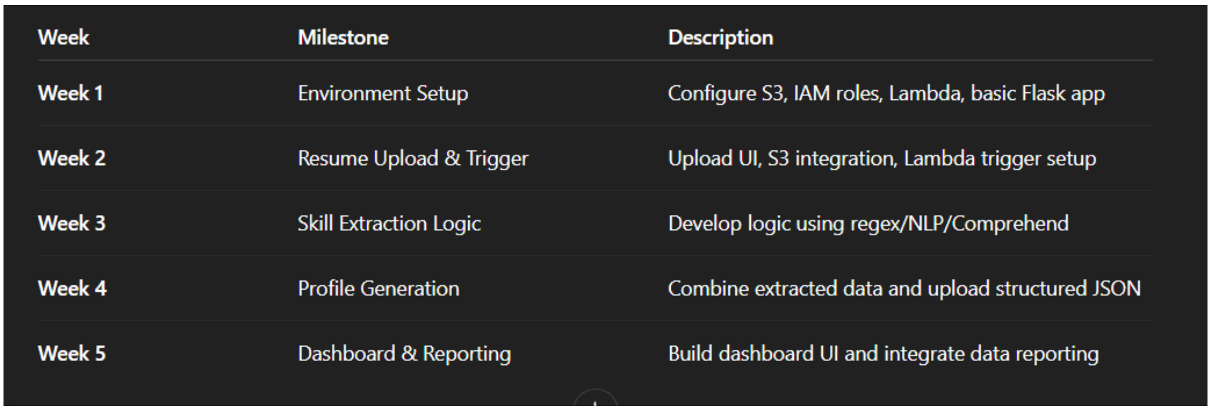
  Output: .json, .csv

  Intermediate: Plain text for processing

3. Project Management

Methodology: Agile methodology with iterative development in 2-week sprints.

Milestones and Deadlines:



Week	Milestone	Description
Week 1	Environment Setup	Configure S3, IAM roles, Lambda, basic Flask app
Week 2	Resume Upload & Trigger	Upload UI, S3 integration, Lambda trigger setup
Week 3	Skill Extraction Logic	Develop logic using regex/NLP/Comprehend
Week 4	Profile Generation	Combine extracted data and upload structured JSON
Week 5	Dashboard & Reporting	Build dashboard UI and integrate data reporting

Team Meeting Workflow

To maintain smooth coordination throughout the *Resume Parser and Skill Extractor* project, we scheduled team meetings on every alternate weekday (excluding weekends). These meetings enabled us to plan, divide work, review progress, and handle any blockers collaboratively.

Sprint 1: Planning Phase (1st July – 11th July)

This sprint focused on requirement gathering, technical planning, team role assignment, and designing the end-to-end architecture.

July 1
Kick-off meeting to introduce the project, divide responsibilities across modules, and finalize the tech stack. Initial project architecture and sprint goals were discussed.

July 3
We focused on setting up the resume upload UI and discussed the S3 bucket configuration to store PDF files uploaded by users.

July 7
Discussed Lambda function logic and how it would parse PDF resumes. Team members were assigned sub-tasks around parsing and handling different resume formats.

July 8
Decided the schema for storing parsed JSON data in DynamoDB. Discussed edge cases like missing fields and how to handle such resumes.

July 11
Sprint 1 wrap-up meeting. Validated the AWS S3 → Lambda → DynamoDB pipeline and ensured planning tasks were completed before moving to development.

Sprint 2: Implementation Phase (14th July – Present)

This sprint focused on development, testing, integration of modules, and preparation of the final dashboard and demo.

July15
Started implementing AWS Comprehend for Named Entity Recognition (NER) to extract meaningful entities like skills, education, and roles from resumes.

July17
Integration of Bedrock/Gemini API for advanced profile summarization. Team reviewed early output quality and began testing.

July21
Initial version of the admin dashboard was completed. Discussed layout for resume listings, filters, skill tags, and dashboard statistics.

July22
Implemented and tested logging via AWS CloudWatch. Added CSV export functionality for candidate data from the dashboard.

July25
End-to-end testing of the entire system: upload → parse → store → visualize. UI refinements and bug fixes were discussed and distributed.

July26
Code freeze preparation and final checklist. Dashboard visuals, theme toggling, and candidate detail pages were polished.

July29
Conducted full demo rehearsal. Ensured all modules worked seamlessly and discussed minor UI/UX improvements before final submission.

July31
Final team sync-up before project submission. Documentation review, success rate metrics, and last bug patches were discussed.

Technology Stack and Usage

1.Amazon S3(Simple Storage Service):

Purpose: Acts as the primary storage for all uploaded resume files (.pdf)

Used In: Upload_UI/app.py and parsing_module/lambda_function.py

How it works:

- Users upload resumes via a Flask-based frontend.
- The file is sent to a configured S3 bucket using the boto3 client.
- The uploaded file triggers an AWS Lambda function automatically (event-based trigger).

2. AWS Lambda:

Purpose: Serverless compute service used to parse resumes and generate structured JSON.

Used In: parsing_module folder, especially lambda_function.py

How it works:

- Triggered automatically when a new file is uploaded to S3.
- Fetches the file from S3.
- Reads file content using libraries like pdfminer.
- Passes extracted text to parsing functions, NER (Comprehend), and LLM summarizers.
- Final structured JSON is uploaded to DynamoDB.

3.Amazon DynamoDB:

Purpose: NoSQL database used to store structured candidate profiles in JSON format.

Used In: lambda_function.py for storing data, Analytics_dashboard/main.py for retrieving data.

How it works:

- Lambda inserts the extracted JSON into a DynamoDB table with candidate details.

- Dashboard queries DynamoDB to display analytics like skill counts, experience summaries, etc.

4.AWS Comprehend:

Purpose: Performs Named Entity Recognition (NER) on resume text to extract key elements like names, dates, locations, job titles, etc.

Used In: lambda_function.py

How it works:

- Text extracted from resume is passed to Comprehend.
- Comprehend returns detected entities with confidence scores.
- These entities help populate the structured profile JSON more accurately.

5.Google Gemini API:

Purpose: Provides LLM-based summarization of candidate experience and job responsibilities.

Used In: summarize_experience.py

Why Gemini?

- Chosen over AWS Bedrock in this project.
- Offers powerful summarization via Google's LLM models.

How it works:

- Relevant text from the experience section is passed to Gemini.
- Gemini returns a concise, readable summary.

Integration: Via HTTP API requests (likely using requests or Google's SDK)

6.AWS CloudWatch:

Purpose: For monitoring Lambda function executions, logging errors, and tracking performance.

Used In: Not directly coded, but implicitly used by Lambda.

How it works:

- Every Lambda invocation automatically logs to CloudWatch.
- Errors, start/stop times, memory usage, etc., can be tracked via AWS Console.

7.Flask:

Purpose: Lightweight web framework used to build:

- Resume Upload Interface (Upload_UI)
- Analytics Dashboard (Analytics_dashboard)

Used In: app.py, main.py

How it works:

- Upload UI allows file submission using a form.
- Dashboard fetches data from DynamoDB and visualizes it.

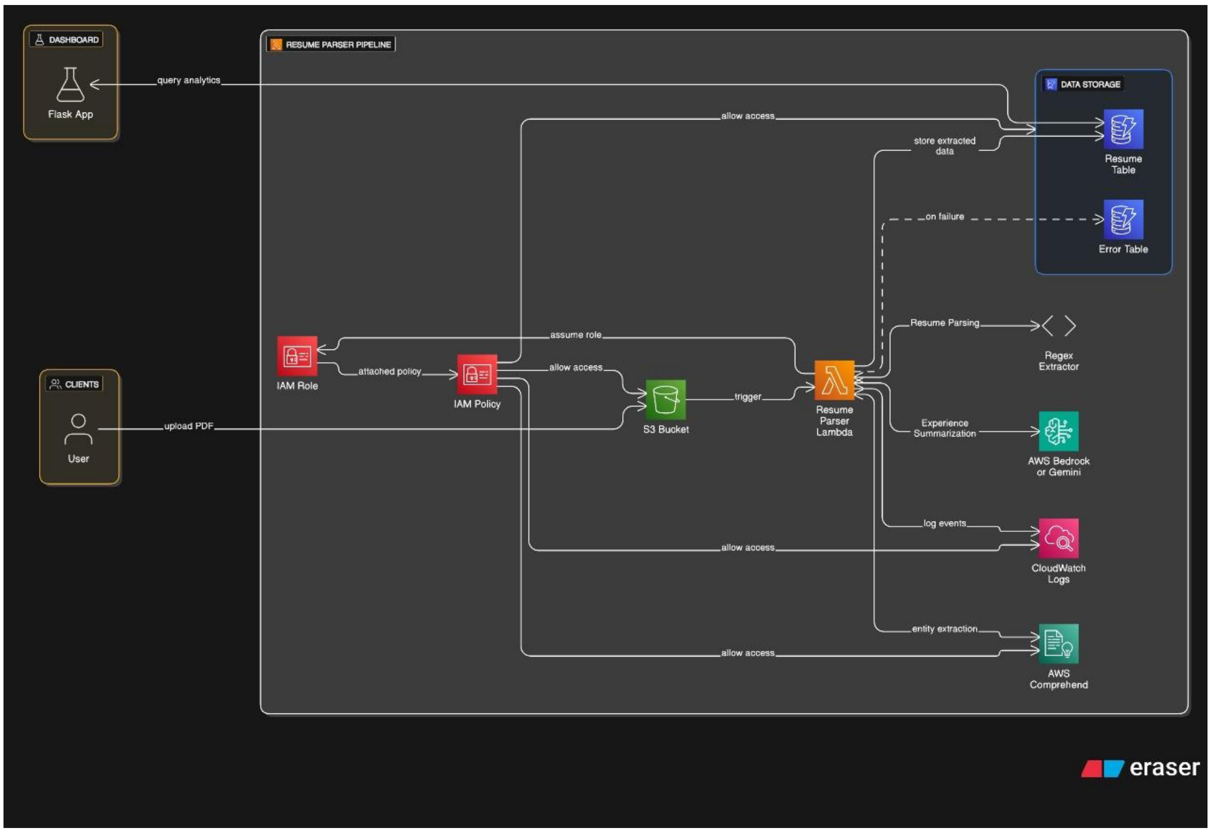
8.Boto3:

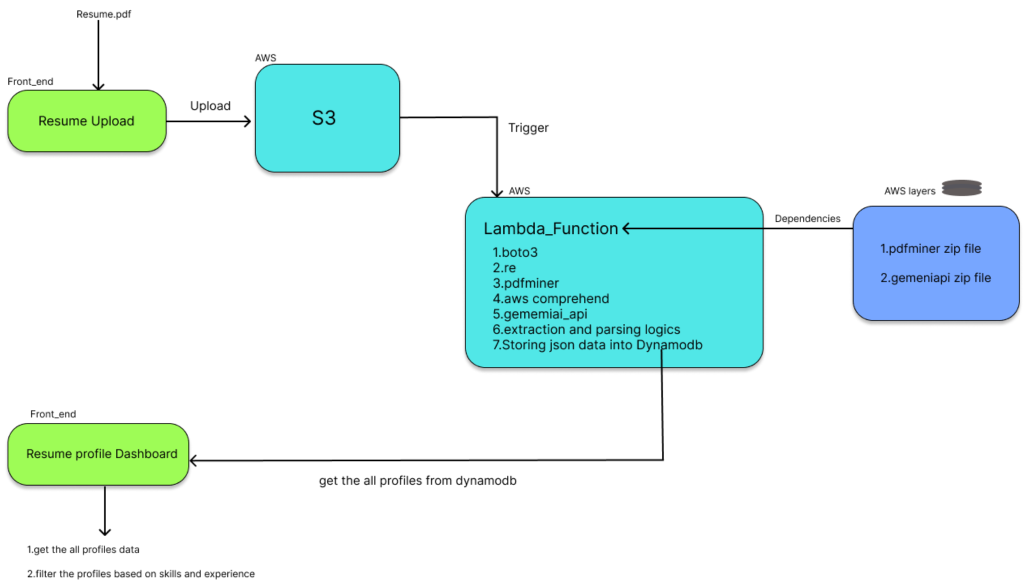
Purpose: Python SDK to interact with AWS services (S3, DynamoDB, Comprehend, etc.)

Used In: Across all backend modules

Why it's important: It abstracts the complexity of REST APIs and offers easy-to-use Python methods.

System Architecture





The system is an automated, cloud-native resume parsing pipeline designed to programmatically extract, analyze, and store key information from candidate resumes. The architecture is built entirely on Amazon Web Services (AWS), employing a serverless, event-driven model to ensure scalability, cost-efficiency, and maintainability.

The workflow can be broken down into the following key stages:

1. Data Ingestion

The process initiates when a **client** uploads a resume, typically a PDF document, into a designated **Amazon S3 bucket**. S3 serves as the primary, highly-durable storage layer for the raw source files. The access for this upload is managed securely through pre-defined permissions.

2. Processing Core

The upload of a new object to the S3 bucket acts as an event trigger for the central processing unit of the pipeline: an **AWS Lambda function** named "Resume Parser Lambda". This function is responsible for orchestrating the entire data extraction and analysis workflow by integrating with several specialized AI and NLP services:

- **Experience Summarization:** For understanding and condensing the unstructured text of a candidate's work history, the Lambda function interfaces with a Large Language Model (LLM) through **AWS Bedrock or Google's Gemini**. This generates a concise summary of the candidate's experience.
- **Named Entity Recognition:** The function leverages **AWS Comprehend**, a natural language processing service, to perform entity extraction. This accurately identifies and categorizes specific pieces of information such as skills, company names, job titles, and dates.
- **Structured Data Parsing:** A **Regex (Regular Expression) Extractor** is used to parse well-defined patterns like email addresses and phone numbers with high precision.

3. Data Storage and Error Handling

Once the resume is processed, the structured, extracted data is persisted in a **Resume Table** within the system's primary data storage. To ensure the pipeline is robust, a failure-handling mechanism is in place. If the Lambda function encounters an error during processing, the failure details are logged to a separate **Error Table**, which aids in debugging and system monitoring.

4. Monitoring and Analytics

- **Logging:** All operational events, execution logs, and potential errors from the Lambda function are sent to **Amazon CloudWatch Logs**. This provides comprehensive monitoring and observability for the health and performance of the pipeline.

- **Dashboard:** A **Flask web application** serves as the user-facing **Dashboard**. It provides an interface for users (e.g., recruiters or hiring managers) to query the analytics and view the extracted candidate information stored in the Resume Table.

5. Security and Access Control

Security is managed through **AWS Identity and Access Management (IAM)**. The entire workflow is governed by an **IAM Policy** attached to an **IAM Role**. The Resume Parser Lambda function **assumes** this role to gain the necessary, limited permissions ("allow access") to interact with other AWS resources, including the S3 bucket, AWS Comprehend, CloudWatch Logs, and the data storage tables. This follows the principle of least privilege, ensuring that each component only has access to the resources essential for its function.

Challenges Faced: -

1. Text Noise and Complex PDF Layouts

- **Problem/Error**:
  - Extracted text by using **Pdfminer.Six,** but it had unwanted characters, broken lines, headers/footers unwanted spaces.
  - PDF contained **tables**, **multi-column layouts**, and graphical content.
- **Solution**:
  - Applied **custom regex cleaning rules** to remove line breaks, symbols, and repetitive headers/footers.

2. S3 Permission and IAM Role Issues

- **Problem/Error**:
  - Lambda wasn't triggered, or couldn't access files from S3.
  - Access denied or permission errors occurred during testing.
- **Solution**:
  - Updated **S3 bucket policies** to allow Lambda execution.
  - Attached **correct IAM role** with permissions: s3:Fullobjectaccess and lambda:InvokeFunction.
  - Enabled detailed **CloudWatch logs** to trace permission issues.

3. spaCy Layer Size Limit in Lambda

- **Problem/Error**:
  - spaCy model (en_core_web_sm) exceeded **100+ MB**, but Lambda **layer zipped size limit is 50 MB**.
  - Couldn't upload or attach the model in AWS Lambda.
- **Solution**:
  - **Removed spaCy dependency**.
  - Used **custom skill keyword dictionary** and **regex pattern matching** for skill extraction.
  - Categorized skills manually (Programming, Tools, Soft Skills).

4. Lambda Memory and Timeout Limits

- **Problem/Error**:
  - Lambda failed or timed out when processing large PDFs.
  - Default memory (128 MB) and timeout (3 seconds) were insufficient.
- **Solution**:
  - **Increased memory allocation** to 512 MB and **timeout** to 60 seconds.
  - Logged processing time and optimized heavy operations (e.g., using fitz from PyMuPDF efficiently).

5. Unreadable or Corrupted PDFs

- **Problem/Error**:
  - Some uploaded resumes were **image-based** or **damaged**, failing during parsing.
  - Resulted in null or empty extracted text.
- **Solution**:
  - Added **error handling and try-except blocks** in the parsing function.
  - Logged file names and errors into CloudWatch.
  - Flagged resumes as **"Parsing error rate"** in output for admin review.

6. Difficulty in Extracting Name from Resume

- **Problem/Error**:
  - Candidate names were not consistently extractable due to **different resume formats** (e.g., name in header, inside a paragraph, capitalized, or styled differently).
  - Regex or keyword-based rules were not reliable in identifying names.
- **Solution**:
  - Integrated **AWS Comprehend** for **Named Entity Recognition (NER)** to extract names with high accuracy.
  - Used entity type "PERSON" from Comprehend's output to confidently extract candidate names regardless of format or placement.

7. Inconsistent Extraction of Experience (Years, Company, Domain)

- **Problem/Error**:
  - Extracting **years of experience**, **company names**, and **domain/role** was inaccurate using simple keyword methods.
  - Some resumes had informal or narrative styles, making regex parsing unreliable.
- **Solution**:
  - Used **Gemini API** (LLM-based) to analyze full text and extract structured information:
    - Total experience duration
    - Company names mentioned
    - Roles/domains worked in

Output Images – UI Workflow Screenshots

1. Landing Page – Resume Upload UI

- This is the main landing page where users can upload their resumes in PDF format.
- Users can drag & drop or click to select a file for upload.
- A clean AWS-branded UI ensures a smooth and intuitive user experience.

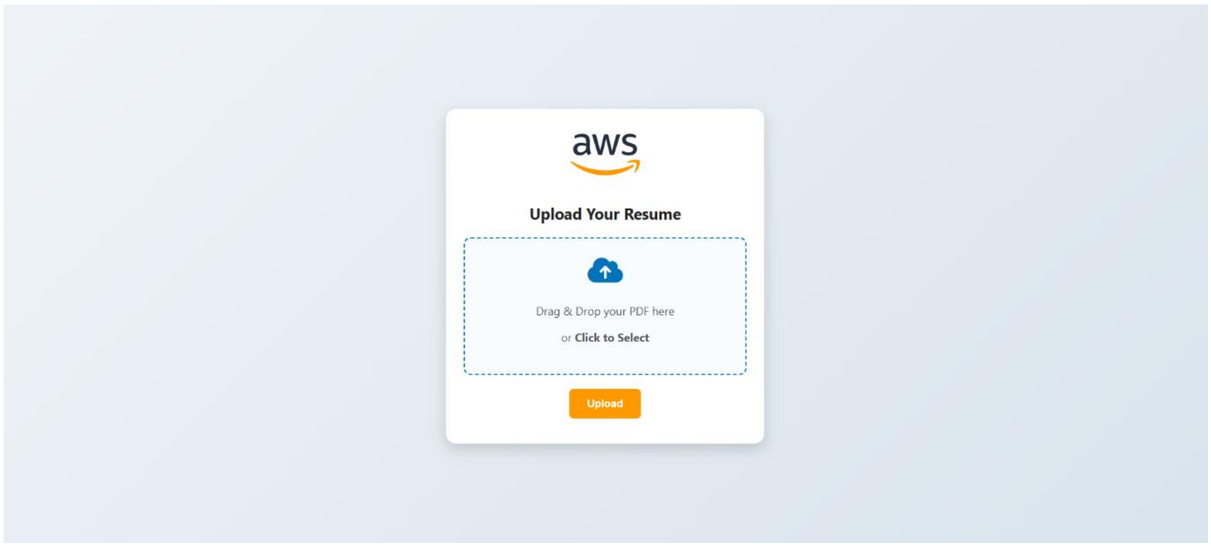
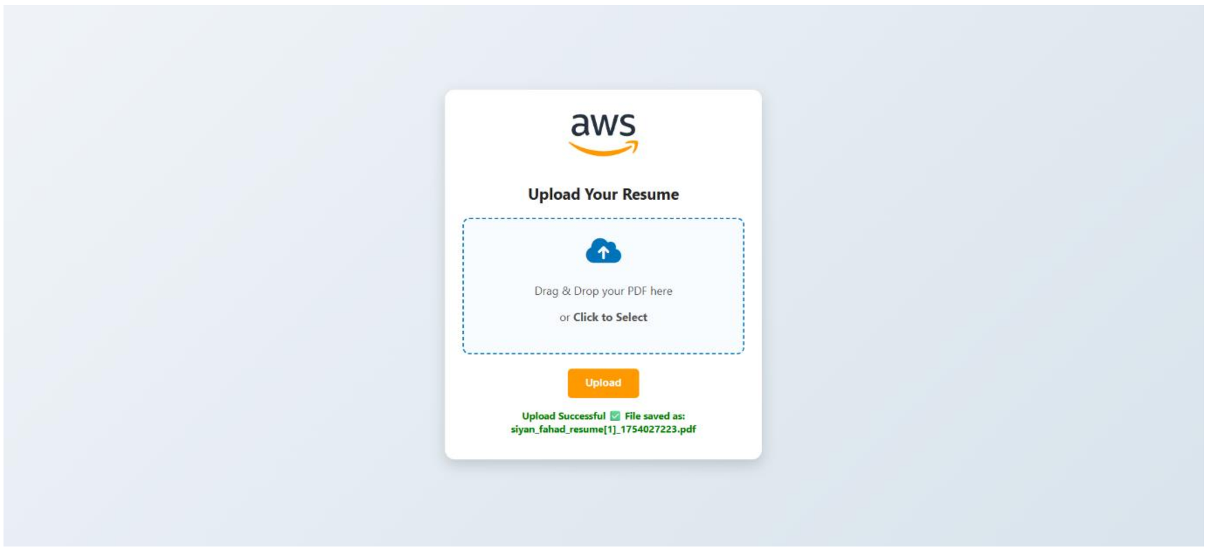


fig 1: Landing Page – Resume Upload UI

2. Upload Acknowledgement – Success Message on UI

- Once a resume is uploaded, a confirmation message is shown on the same upload UI.
- The message includes a success checkmark and the saved filename.
- This confirms to the user that their resume has been successfully sent to the server.

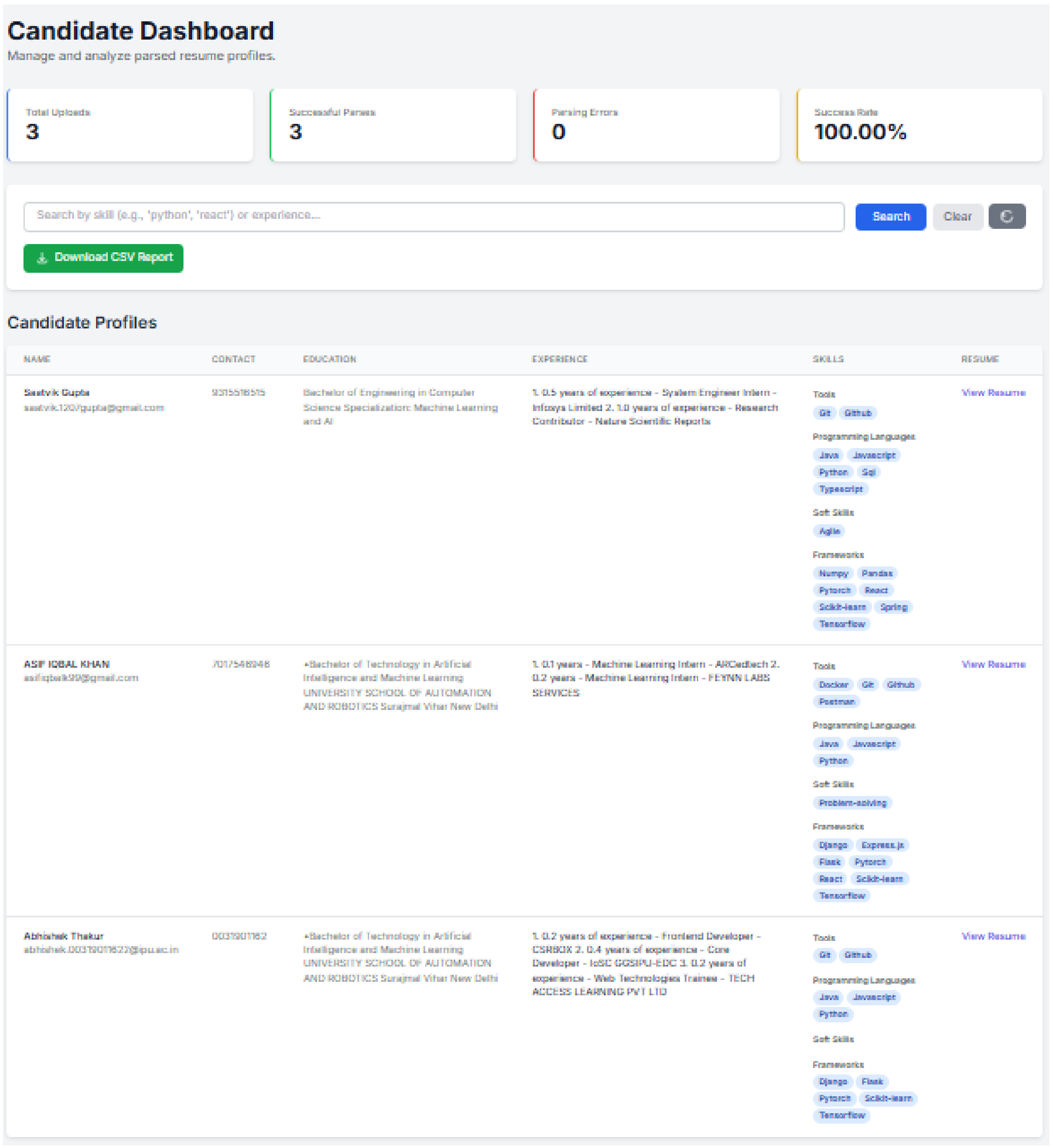


Upload Successful ✅ File saved as: siyan_fahad_resume[1]_1754027223.pdf

Fig 2: ***Upload Acknowledgement – Success Message on UI***

3. Admin Dashboard – Before Refresh (3 Resumes)

- The admin dashboard initially displays 3 parsed resume entries.
- Each row includes candidate info like name, contact, education, experience, skills, and a link to view the resume.
- Stats at the top summarize total uploads, successful parses, and success rate.



Candidate Dashboard

Manage and analyze parsed resume profiles.

Total Uploads	Successful Parses	Parsing Errors	Success Rate
3	3	0	100.00%

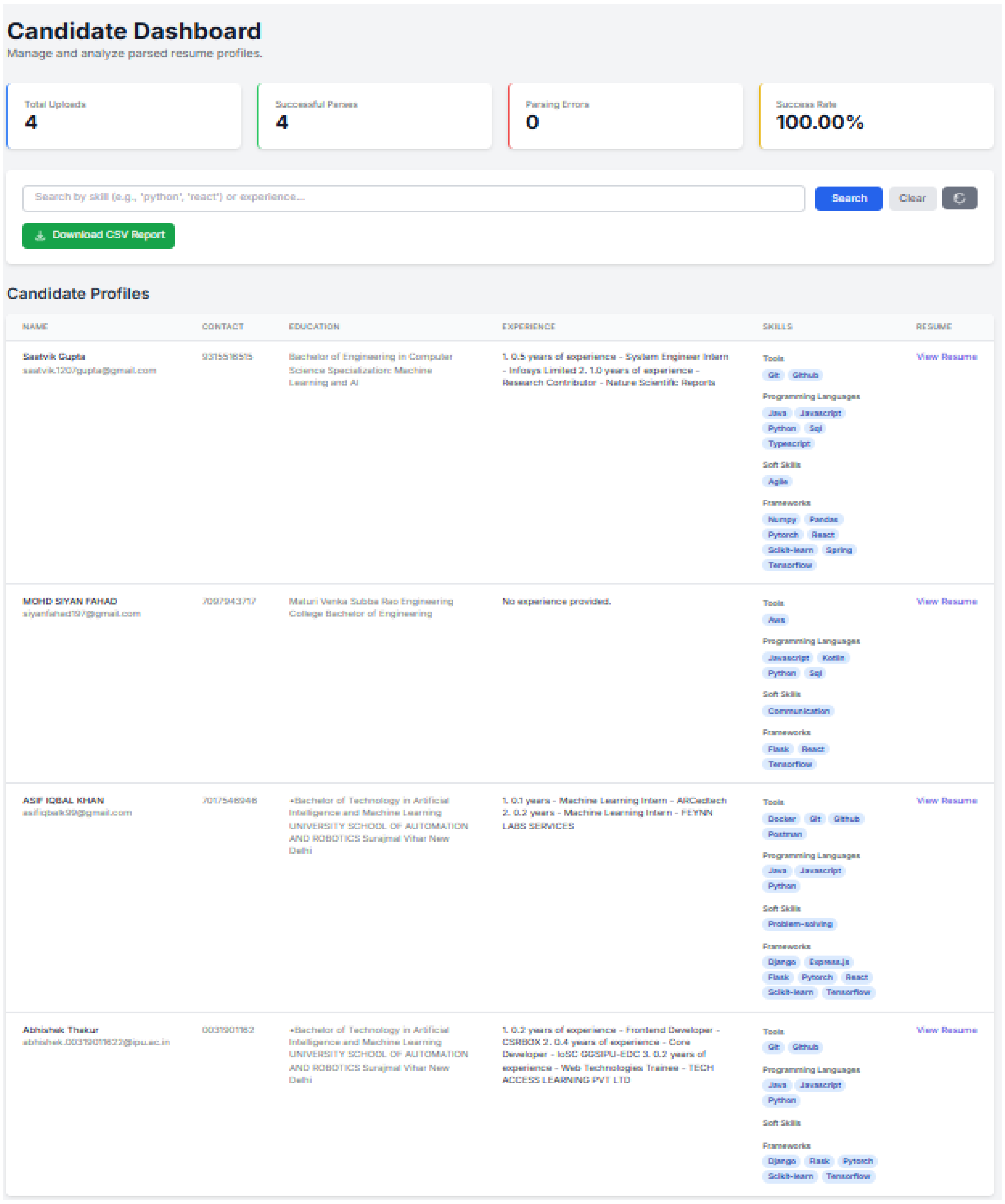
Candidate Profiles

NAME	CONTACT	EDUCATION	EXPERIENCE	SKILLS	RESUME
Saatvik Gupta saatvik.1207gupta@gmail.com	9315518515	Bachelor of Engineering in Computer Science Specialization: Machine Learning and AI	1. 0.5 years of experience - System Engineer Intern - Infosys Limited 2. 1.0 years of experience - Research Contributor - Nature Scientific Reports	Tools: Git, Github; Programming Languages: Java, Javascript, Python, Sql, Typescript; Soft Skills: Agile; Frameworks: Numpy, Pandas, Pytorch, React, Scikit-learn, Spring, Tensorflow	View Resume
ASIF IQBAL KHAN asifiqbalk99@gmail.com	7017548948	•Bachelor of Technology in Artificial Intelligence and Machine Learning UNIVERSITY SCHOOL OF AUTOMATION AND ROBOTICS Surajmal Vihar New Delhi	1. 0.1 years - Machine Learning Intern - ARCedtech 2. 0.2 years - Machine Learning Intern - FEYNN LABS SERVICES	Tools: Docker, Git, Github, Postman; Programming Languages: Java, Javascript, Python; Soft Skills: Problem-solving; Frameworks: Django, Express.js, Flask, Pytorch, React, Scikit-learn, Tensorflow	View Resume
Abhishek Thakur abhishek.00319011622@ipu.ac.in	0031901162	•Bachelor of Technology in Artificial Intelligence and Machine Learning UNIVERSITY SCHOOL OF AUTOMATION AND ROBOTICS Surajmal Vihar New Delhi	1. 0.2 years of experience - Frontend Developer - CSRBOX 2. 0.4 years of experience - Core Developer - IoSC GGSIPU-EDC 3. 0.2 years of experience - Web Technologies Trainee - TECH ACCESS LEARNING PVT LTD	Tools: Git, Github; Programming Languages: Java, Javascript, Python; Soft Skills: ; Frameworks: Django, Flask, Pytorch, Scikit-learn, Tensorflow	View Resume

Fig 3: Admin Dashboard – Before Refresh

4. Admin Dashboard – After Refresh

- After refreshing the dashboard, the newly uploaded resume appears, updating the count to 4.
- This demonstrates real-time integration of the upload system with the backend parser and database.
- The success rate remains 100%, indicating all resumes parsed without errors.



Candidate Dashboard

Manage and analyze parsed resume profiles.

Total Uploads	Successful Parses	Parsing Errors	Success Rate
4	4	0	100.00%

Candidate Profiles

NAME	CONTACT	EDUCATION	EXPERIENCE	SKILLS	RESUME
Saatvik Gupta saatvik.1207gupta@gmail.com	9315518515	Bachelor of Engineering in Computer Science Specialization: Machine Learning and AI	1. 0.5 years of experience - System Engineer Intern - Infosys Limited 2. 1.0 years of experience - Research Contributor - Nature Scientific Reports	Tools: Git, Github; Programming Languages: Java, Javascript, Python, Sql, Typescript; Soft Skills: Agile; Frameworks: Numpy, Pandas, Pytorch, React, Scikit-learn, Spring, Tensorflow	View Resume
MOHD SIYAN FAHAD siyanfahad197@gmail.com	7097943717	Maturi Venka Subba Rao Engineering College Bachelor of Engineering	No experience provided.	Tools: Aws; Programming Languages: Javascript, Kotlin, Python, Sql; Soft Skills: Communication; Frameworks: Flask, React, Tensorflow	View Resume
ASIF IQBAL KHAN asifiqbalk99@gmail.com	7017548948	•Bachelor of Technology in Artificial Intelligence and Machine Learning UNIVERSITY SCHOOL OF AUTOMATION AND ROBOTICS Surajmal Vihar New Delhi	1. 0.1 years - Machine Learning Intern - ARCedtech 2. 0.2 years - Machine Learning Intern - FEYNN LABS SERVICES	Tools: Docker, Git, Github, Postman; Programming Languages: Java, Javascript, Python; Soft Skills: Problem-solving; Frameworks: Django, Express.js, Flask, Pytorch, React, Scikit-learn, Tensorflow	View Resume
Abhishek Thakur abhishek.00319011822@ipu.ac.in	0031901182	•Bachelor of Technology in Artificial Intelligence and Machine Learning UNIVERSITY SCHOOL OF AUTOMATION AND ROBOTICS Surajmal Vihar New Delhi	1. 0.2 years of experience - Frontend Developer - CSRBOX 2. 0.4 years of experience - Core Developer - IoSC GGSIPU-EDC 3. 0.2 years of experience - Web Technologies Trainee - TECH ACCESS LEARNING PVT LTD	Tools: Git, Github; Programming Languages: Java, Javascript, Python; Soft Skills: ; Frameworks: Django, Flask, Pytorch, Scikit-learn, Tensorflow	View Resume

Fig 4: Admin Dashboard – After Refresh

Conclusion:

The Resume Parser and Skill Extractor project successfully demonstrates how cloud-based automation and natural language processing (NLP) can streamline the resume screening process. By integrating AWS services such as S3 and Lambda with Python libraries like PyMuPDF, pdfminer.six the system eliminates manual effort, reduces human error, and enhances the speed and accuracy of candidate evaluation.

The modular approach—ranging from resume upload to profile generation and dashboard reporting—ensures flexibility, scalability, and ease of maintenance. The optional web-based dashboard adds value by allowing recruiters or administrators to search, filter, and download reports, making the system not only intelligent but also user-friendly.

This project proves to be a robust solution for recruitment teams, HR departments, and educational institutions by enabling data-driven hiring decisions. With further enhancements such as AI-based ranking, candidate similarity scoring, or multi-language support, this solution can evolve into a comprehensive talent acquisition platform