Future Job Notifications

with RChilli

June 2024 | Version 1.0

Purpose Statement

This document provides a detailed guide for integrating the **Future Job Notifications** use case using RChilli Search Engine APIs. The feature matches new job postings with indexed candidate profiles and generates automated job notification alerts to the matched candidates.

Disclaimer:

Contents

Introduction

The **Future Job Notifications** feature aims to enhance candidate engagement by alerting them about new job opportunities that match their profiles. Utilizing RChilli's Search Engine APIs, the system automates the parsing and indexing of candidate resumes and job descriptions, enabling accurate matching and timely notifications.

RChilli Search Engine is a design for searching, matching and ranking needs, and it utilizes IndexIds to relate information stored in your database with same Ids to generate complete information.

Key Components and Flow

1. Parse and Index Candidates

- **Action:** Upload, parse, and index candidates' resumes using the RChilli Search Engine API.
- **Output:** Parsed JSON data and indexing confirmation.

2. Store Candidate Information

- **Action:** Store parsed candidate data in your database for application use and future matching based on RChilli index IDs.
- **Output:** Candidate profiles saved in the database.

3. Parse and Index Jobs

- **Action:** Parse and index new job descriptions using the RChilli Search Engine API.
- **Output:** Structured JSON data of job postings, indexed for database storage.

4. Store Job Information

- **Action:** Store parsed job data in your database for application use and future matching based on RChilli index IDs.
- **Output:** Job information saved in the database.

5. Match Jobs with Candidates

- **Action:** Compare new job postings with indexed candidate profiles to identify potential matches, adjust dynamic weightage-based business need to get best candidate ranking.
- **Output:** List of matching candidates with relevance scores.

6. Generate Job Notifications

- **Action:** Notify candidates about relevant job postings via email or in-app notifications.

RChilli Match returns the candidate with score, you need define your criteria for notifications.

- **Output:** Automated job alerts sent to candidates.

Benefits

- **Improved Candidate Engagement:** Keeps candidates informed about new opportunities.
- **Efficient Matching:** Leverages AI to match job postings with candidate profiles accurately.
- **Automated Alerts:** Reduces manual effort in notifying candidates about new job openings.

Example Scenario

- **Scenario**: Alex, a candidate interested in Data Science roles, has uploaded their profile to the job portal.
- **Action**: A recruiter uploads a new job posting for a Data Scientist position to the platform.
- **Response**:
    - The system processes the job description using the RChilli Search and Match Engine API.
    - It extracts key information such as required skills, experience, and qualifications and Index it RChilli Search and Match Engine.
    - The system then matches this data against indexed candidate profiles using RChilli Search and Match Engine API.
    - Candidate with Data Scientist profiles match the job criteria based on their skills, experience, and education.
- **Outcome**: Alex receives a notification via email and the job portal about the new Data Scientist position. Alex logs into the job portal, reviews the job details, and decides to apply for the position.

User Stories

1. **Parse and Index Resumes**
    - **As a system,** I need to parse and index candidate resumes to prepare them for job matching and notifications.
    - **Acceptance Criteria:**
        - Resumes are parsed into structured JSON format and indexed in the database.
2. **Parse and Index Job Descriptions**
    - **As a system**, I need to parse and index job descriptions to facilitate matching with candidate profiles.
    - **Acceptance Criteria:**
        - Job descriptions are parsed into structured JSON format and indexed in the database.
3. **Match Jobs with Candidates**
    - **As a system**, I want to match new job postings with indexed candidate profiles to identify relevant candidates.
    - **Acceptance Criteria:**
        - Job postings are accurately matched with candidate profiles based on criteria.
4. **Send Job Notifications**
    - **As a candidate,** I want to receive notifications about new job postings that match my profile so that I can apply for relevant jobs.
    - **Acceptance Criteria:**
        - Candidates receive notifications about new matching job postings.

User Acceptance Testing (UAT)

UAT Scenario 1: Parsing and Indexing of Candidate Resumes

- **Objective**: Verify that candidate resumes are parsed and indexed successfully.
- **Preconditions**: Candidate has uploaded a resume.
- **Steps**:
    1. Upload a resume.
    2. Parse and index the resume.
- **Expected Result**: Resume data is parsed and indexed without errors.
- **Acceptance Criteria**:
    - Structured data is generated accurately.

UAT Scenario 2: Parsing and Indexing of New Job Descriptions

- **Objective**: Verify that new job descriptions are parsed and indexed successfully.
- **Preconditions**: New job descriptions are available.
- **Steps**:
    1. Add a new job description to the system.
    2. Parse and index the job description.
- **Expected Result**: Job description data is parsed and indexed without errors.
- **Acceptance Criteria**:
    - Structured job data is generated accurately.

UAT Scenario 3: Accurate Matching of New Jobs with Candidate Profiles

- **Objective**: Confirm that new job descriptions are accurately matched with candidate profiles.
- **Preconditions**: Parsed and indexed candidate profiles and job descriptions.
- **Steps**:
    1. Compare new job descriptions with candidate profiles.
    2. Generate a list of matching jobs.
- **Expected Result**: The system identifies relevant job matches.

- **Acceptance Criteria**:
    - Job matches are relevant to the candidates' profiles.

UAT Scenario 4: Job Notifications Generation and Delivery

- **Objective**: Verify that job notifications are correctly generated and sent to candidates.
- **Preconditions**: Matching jobs identified for candidates.
- **Steps**:
    1. Generate job notifications for candidates.
    2. Send notifications via email, SMS, or in-app alerts.
- **Expected Result**: Notifications are sent and received accurately.
- **Acceptance Criteria**:
    - Candidates receive notifications with relevant job details.

Test Cases

Test Case 1: Candidate Resume Parsing Test

- **Objective**: Ensure candidate resumes are parsed and indexed successfully.
- **Preconditions**: Candidate uploads a resume.
- **Steps**:
    1. Upload a resume.
    2. Parse the resume using the Resume Parser API.
    3. Index the parsed data.
- **Expected Result**: Resume data is correctly parsed and indexed.
- **Pass Criteria**: JSON data structure accurately represents the resume content.

Test Case 2: New Job Description Parsing Test

- **Objective**: Ensure new job descriptions are parsed and indexed successfully.
- **Preconditions**: New job description provided by the recruiter.
- **Steps**:

1. Add a new job description to the system.
2. Parse the description using the Job Parser API.
3. Index the parsed data.

- **Expected Result**: Job data is correctly parsed and indexed.
- **Pass Criteria**: JSON data structure accurately represents the job description.

Test Case 3: Matching New Jobs with Candidate Profiles Test

- **Objective**: Validate that the system accurately matches new job descriptions with candidate profiles.
- **Preconditions**: Parsed candidate profiles and job descriptions available.
- **Steps**:
    1. Use matching algorithms to compare new job descriptions with candidate profiles.
    2. Generate job matches.
- **Expected Result**: The system identifies and matches relevant new jobs with candidates.
- **Pass Criteria**: Recommendations are relevant to the candidate profiles.

Test Case 4: Job Notification Generation Test

- **Objective**: Ensure job notifications are generated for candidates based on new job matches.
- **Preconditions**: Matching jobs identified for candidates.
- **Steps**:
    1. Generate job notifications.
    2. Verify the generated notification details.
    3. Verify Duplicate job notification shouldn’t be sent for same job.
- **Expected Result**: Notifications contain relevant job details.
- **Pass Criteria**: Notifications are correctly generated with accurate information.

Test Case 5: Job Notification Delivery Test

- **Objective**: Verify that job notifications are delivered to candidates via various channels.
- **Preconditions**: Job notifications generated.

- **Steps**:
    1. Send notifications via email, SMS, or in-app alerts.
    2. Verify receipt by candidates.
- **Expected Result**: Notifications are received by candidates on time and accurately.
- **Pass Criteria**: Notifications are delivered without delay or error.

Test Case 6: Error Handling for Parsing Failures

- **Objective**: Ensure system handles errors in parsing resumes or job descriptions gracefully.
- **Preconditions**: Invalid or poorly formatted resume or job description.
- **Steps**:
    1. Upload a resume or job description with errors.
    2. Attempt to parse the document.
- **Expected Result**: System provides meaningful error messages.
- **Pass Criteria**: System handles errors without crashing and provides clear feedback.

Test Case 7: Data Storage Verification for Resumes and Jobs

- **Objective**: Verify that parsed resume and job data are correctly stored in the database.
- **Preconditions**: Parsed data available.
- **Steps**:
    1. Parse and index resume and job descriptions.
    2. Check the database for stored data.
- **Expected Result**: Data is stored accurately and retrievably.
- **Pass Criteria**: Database entries reflect parsed data correctly.

Test Case 8: Notification Preferences Test

- **Objective**: Ensure candidates can set and receive notifications according to their preferences.
- **Preconditions**: Candidates have preferences for notification channels.
- **Steps**:

1. Set notification preferences (email, SMS, in-app).
2. Generate and send notifications.

- **Expected Result**: Notifications are sent according to the set preferences.
- **Pass Criteria**: Notifications match candidate preferences.

Test Case 9: Performance Under High Volume

- **Objective**: Ensure the system handles high volumes of new job postings and candidate matches without performance issues.
- **Preconditions**: High volume of new job postings.
- **Steps**:
    1. Simulate adding multiple new job descriptions.
    2. Monitor system performance.
- **Expected Result**: System handles the load efficiently.
- **Pass Criteria**: System remains responsive and accurate.

Test Case 10: Duplicate Resume Handling

- **Objective**: Ensure that the system handles duplicate resumes without re-parsing or re-indexing the same resume multiple times.
- **Preconditions:** Candidate uploads a resume that is already present in the database.
- **Steps:**
    1. Upload a duplicate resume.
    2. Attempt to parse and index the resume.
- **Expected Result:** The system detects the duplicate resume and does not re-parse or re-index it.
- **Pass Criteria:**
    1. The system identifies duplicate resumes and avoids unnecessary processing.
    2. Candidates who have opted out do not receive any job notifications.

Test Case 11: Multi-Language Support for Resume Parsing

- **Objective:** Verify that resumes in multiple languages are parsed correctly.

- **Preconditions:** Resumes in different languages are available for upload.
- **Steps:**
    1. Upload resumes in various languages.
    2. Parse the resumes using the Resume Parser API.
    3. Verify the parsed data.
- **Expected Result:** Resumes are parsed accurately regardless of language.
- **Pass Criteria:** Parsed data accurately represents the content of the resumes in different languages.

Test Case 12: Error Handling for API Failures

- **Objective:** Ensure the system handles API failures gracefully and provides meaningful error messages.
- **Preconditions:** Simulate API failure conditions.
- **Steps:**
    1. Simulate an API failure (e.g., by disconnecting the network or providing invalid API keys).
    2. Attempt to parse and index resumes or job descriptions.
- **Expected Result:** System handles the API failure gracefully and provides clear error messages.
- **Pass Criteria:** System does not crash and provides meaningful error messages during API failures.

Test Case 13: Data Security and Privacy

- **Objective:** Ensure that candidate and job data are stored securely, and privacy is maintained.
- **Preconditions:** Parsed data is available in the database.
- **Steps:**
    1. Check the database for security measures (e.g., encryption, access controls).
    2. Verify that sensitive data is stored securely.
- **Expected Result:** Data is stored securely with appropriate privacy measures.

- **Pass Criteria:** Database entries are secure and comply with privacy regulations.

Test Case 14: User Interface Responsiveness

- **Objective:** Ensure that the user interface remains responsive during high-volume operations. Preconditions: High volume of data processing (resumes and job descriptions).
- **Steps:**
    1. Perform high-volume operations (upload, parse, and index resumes and job descriptions).
    2. Interact with the user interface during these operations.
- **Expected Result:** User interface remains responsive and functional.
- **Pass Criteria:** UI remains responsive and allows for smooth interaction during high-volume operations.

Miscellaneous

The integration depends on your application workflow. There might be different steps involved in your integration which may vary from application to application.

This document indicates basic steps which are involved, you can review your application and add other UAT, and test cases based upon your needs.

Integration Estimation

Disclaimer: This estimation is based on a sample application with simple workflow. This may vary depending on the complexity of your application and the skill set/experience of the team involved in the integration.

Development (15-18 hrs):

- Resume Parsing and Indexing: 3-4 hrs
- Job Parsing and Indexing: 3-4 hrs
- Matching Algorithm Implementation: 4-5 hrs
- Notification System Integration: 3-4 hrs
- Unit Testing: 2-3 hrs

QA (6-8 hrs):

- UAT Execution: 2-3 hrs
- Test Case Execution: 4-5 hrs

Appendix

Search and Match Api Details:

https://docs.rchilli.com/kc/c_Search_Match_overview

Search and Match Parse and Index API Details:

https://docs.rchilli.com/kc/c_RChilli_search_match_API_Endpoints_Parse_index

Search and Match, Match API Details:

https://docs.rchilli.com/kc/c_RChilli_search_match_API_Endpoints_match

Get API key and Endpoints:

https://docs.rchilli.com/kc/c_RChilli_search_match_API_authentication

Postman Integration and Sample Code:

https://documenter.getpostman.com/view/6003669/Szmh1GQX?version=latest#0b611a07-dd2d-4737-8b0c-68b14d17746f

Search and Match Response Example:

https://docs.rchilli.com/kc/c_RChilli_search_match_Response_example

Search Engine Dynamic Weightage settings:

https://docs.rchilli.com/kc/c_RChilli_search_match_Features_dynamic_weightage_searching

Search and Match API Error Code:

https://docs.rchilli.com/kc/c_RChilli_search_match_error_code

Language Supported:

https://www.rchilli.com/languages