Job Recommendation

with RChilli

June 2024 | Version 1.0

Purpose Statement

This document details integrating the **Job Recommendation** use case leveraging RChilli Search and Match functionality to provide candidates with **job recommendations** matching their profiles.

Contents

Introduction

The **Job Recommendation** feature enhances the job search process by automatically matching candidate profiles to job openings using the RChilli Search and Match API. This automation significantly reduces the time and effort required for candidates to find suitable job opportunities.

Job Recommendation simplifies the job search process by:

- **Uploading Candidate Profiles:** Candidates upload their profiles to the system.
- **Parsing Candidate Profiles:** Profiles are parsed and indexed using the RChilli Search Engin API.
- **Matching Profiles to Jobs:** The system match candidate profiles to job openings using the RChilli Search and Match API.
- **Providing Recommendations:** Candidates receive a list of recommended jobs matching their profiles.

Key Components and Flow

1. **Creating a Jobs Pool:**
    a. **Action:** The application will keep indexing all jobs using the RChilli Parse and Index Method.
    b. **Response:** Generates a structured JSON output along with an index ID for each job.
2. **Store Index ID Reference to Jobs:**
    a. **Action:** Link the job data stored in the database to the index ID for future use.
    b. **Response:** A reference is created for the stored and indexed job.
3. **Profile Upload:**
    a. **Action:** Candidates upload their resumes (PDF, DOCX, etc.) to the system.
    b. **Response:** The profiles are sent to the server for processing.
4. **Profile Parsing:**
    a. **Action:** The server calls the RChilli Search and Match Engine API for parsing and indexing the candidate's resume.
    b. **Functionality:** The API processes the resume, extracting key information and index.
    c. **Response:** The API returns this data in a structured JSON format and stores relevant information with reference to the index ID for future use.
5. **Profile-to-Job Matching:**
    a. **Action:** The system calls the RChilli Search and Match API with the parsed profile data.
    b. **Functionality:** The API matches profiles to job openings based on skills, experience, and education. The matching criteria can be defined dynamically using RChilli's configuration.
    c. **Response:** The API returns a list of recommended jobs for each candidate profile.

6. **Recommendation Display:**
    a. **Action:** The system displays the list of recommended jobs to the candidate with reference to the index ID returned from the Match API and the relevant data stored in the database.
    b. **Response:** Candidates can review, shortlist, and apply to jobs directly from the recommendation list.

Benefits

- **Time Efficiency:** Significantly reduces the time candidates spend on finding job openings that match their profiles.
- **Accuracy:** Enhances the accuracy of job recommendations using RChilli's advanced algorithms.
- **User-Friendly:** Improves the user experience by automating the job recommendation process.

User Stories

1. **Upload Profile**
    - **As a candidate,** I want to upload my profile so that I can receive job recommendations matching my qualifications.
2. **Parse Profile**
    - **As a system,** I need to parse the uploaded profile to extract relevant data and present it in a structured format.
3. **Match Profile to Jobs**
    - **As a system,** I want to match candidate profiles to job openings to provide accurate recommendations.
4. **Display Recommendations**
    - **As a candidate,** I want to see a list of recommended jobs that match my profile.

Example Scenario

1. **Scenario:** A candidate named Alex uploads their profile and wants to find suitable job openings.
2. **Action:** Alex uploads their profile to the system.
3. **Response:** The system processes the profile using the RChilli Resume Parser API, matches the profile to job openings using the RChilli Search and Match API, and displays a list of recommended jobs.
4. **Outcome:** Alex reviews the recommended jobs, shortlists the best ones, and applies to them.

User Acceptance Testing (UAT)

The **Job Recommendation** UAT plan ensures the feature meets user requirements and business goals. It validates that candidates can receive accurate job recommendations by uploading profiles.

UAT Scenarios

UAT Scenario 1: Successful Profile Upload and Parsing

- **Objective:** Verify that a candidate can upload their profile and the system successfully parses it.
- **Preconditions:** Candidate has a profile ready for upload.
- **Steps:**
    1. Navigate to the profile upload page.
    2. Upload a profile.
- **Expected Result:** The profile is uploaded, and the system receives a JSON response with the parsed data.
- **Acceptance Criteria:**
    - Profile upload completes without errors.
    - JSON response includes correctly parsed fields.

UAT Scenario 2: Accurate Profile-to-Job Matching

- **Objective:** Validate that profiles are correctly matched to job openings.
- **Preconditions:** Successful profile parsing.
- **Steps:**
    1. Upload a profile using the system.
    2. Match the profile to job openings using the RChilli API.
- **Expected Result:** Accurate job recommendations are provided.
- **Acceptance Criteria:**
    - Recommendations match the candidate's profile.

UAT Scenario 3: Recommendation Display

- **Objective:** Ensure that job recommendations are displayed correctly.
- **Preconditions:** Successful profile-to-job matching.
- **Steps:**
    1. Upload a profile and match it to job openings.
    2. Display the job recommendations to the candidate.
- **Expected Result:** Job recommendations are displayed accurately.
- **Acceptance Criteria:**
    - Recommendations are relevant and accurate.

UAT Scenario 4: Data Storage Verification

- **Objective:** Confirm that the matched data is correctly stored in the database.
- **Preconditions:** Successful profile parsing and job matching.
- **Steps:**
    1. Upload a profile and match it to job openings.
    2. Verify the stored data in the database.
- **Expected Result:** All matched data is stored accurately and linked to the candidate's profile.
- **Acceptance Criteria:**
    - Database entries match the matched data.
    - Data integrity is maintained.

Test Cases

Test Case 1: Profile Upload Test

- **Objective:** Verify that a profile can be uploaded successfully.
- **Preconditions:** Candidate has a profile ready.
- **Steps:**
    a. Navigate to the profile upload page.
    b. Upload a profile.
- **Expected Result:** Profile upload completes without errors.
- **Pass Criteria:** Profile is uploaded and available for processing.

Test Case 2: API Call Test

- **Objective:** Ensure the profile is parsed successfully by the RChilli API.
- **Preconditions:** Profile is uploaded.
- **Steps:**
    a. Upload a profile.
    b. Verify the API call to RChilli.
    c. Confirm receipt of a JSON response.
- **Expected Result:** JSON response is received with parsed data.
- **Pass Criteria:** API call completes successfully and returns valid JSON.

Test Case 3: Profile-to-Job Matching Test

- **Objective:** Validate that profiles are matched to job openings accurately.
- **Preconditions:** Successful profile parsing.
- **Steps:**
    a. Upload a profile and parse it using the RChilli API.
    b. Match the profile to job openings.
- **Expected Result:** Accurate job recommendations are provided.
- **Pass Criteria:** Recommendations match the candidate's profile.

Test Case 4: Recommendation Display Test

- **Objective:** Confirm that job recommendations are displayed correctly.
- **Preconditions:** Successful profile-to-job matching.
- **Steps:**
    a. Upload a profile and match it to job openings.
    b. Display the job recommendations to the candidate.
- **Expected Result:** Job recommendations are displayed accurately.
- **Pass Criteria:** Recommendations are relevant and accurate.

Test Case 5: Data Storage Test

- **Objective:** Verify that matched data is correctly stored in the database.
- **Preconditions:** Successful profile parsing and job matching.
- **Steps:**
    a. Upload a profile and match it to job openings.
    b. Submit the application.
    c. Verify the stored data in the database.
- **Expected Result:** Matched data is stored accurately in the database.
- **Pass Criteria:** Database entries match the matched profile data.

Test Case 6: Profile Upload Format Validation

- **Objective:** Ensure that the system correctly handles different file formats for profile uploads.
- **Preconditions:** Candidate has profiles in multiple formats (PDF, DOCX, etc.).
- **Steps:**
    a. Navigate to the profile upload page.
    b. Upload profiles in different formats (PDF, DOCX, TXT).
- **Expected Result:** Profiles in all supported formats are uploaded and processed without errors.
- **Pass Criteria:** System accepts and processes profiles in each of the supported formats.

Test Case 7: Invalid Profile Upload Handling

- **Objective:** Verify that the system appropriately handles invalid profile uploads.
- **Preconditions:** Candidate attempts to upload an unsupported or corrupt file.
- **Steps:**
    a. Navigate to the profile upload page.
    b. Attempt to upload an unsupported or corrupt file.
- **Expected Result:** System displays an error message indicating the upload failure.
- **Pass Criteria:** System rejects the invalid file and provides a user-friendly error message.

Test Case 8: Duplicate Profile Upload Detection

- **Objective:** Ensure that the system detects and handles duplicate profile uploads.
- **Preconditions**: Candidate has already uploaded a profile.
- **Steps:**
    a. Upload a profile.
    b. Attempt to upload the same profile again.
- **Expected Result:** System identifies the duplicate upload and prevents it or alerts the user.
- **Pass Criteria:** Duplicate profiles are detected, and appropriate action is taken.

Test Case 9: Profile Parsing Error Handling

- **Objective:** Verify that the system gracefully handles errors during profile parsing.
- **Preconditions:** Candidate uploads a profile that may cause parsing errors (e.g., extremely large file, unusual formatting).
- **Steps:**
    a. Upload a profile that may cause parsing errors.
- **Expected Result:** System handles the error without crashing and provides an appropriate error message to the user.

- **Pass Criteria:** System manages parsing errors and informs the user without affecting the overall functionality.

Test Case 10: Real-Time Profile-to-Job Matching

- **Objective:** Validate that the system performs real-time profile-to-job matching and updates recommendations dynamically.
- **Preconditions:** Profile is uploaded and parsed successfully.
- **Steps:**
    a. Upload a profile.
    b. Modify the profile to include new skills or experiences.
    c. Re-upload the modified profile**.**
- **Expected Result:** System updates the job recommendations in real-time based on the modified profile**.**
- **Pass Criteria:** Recommendations reflect changes in the profile accurately and promptly.

Test Case 11: Profile Deletion and Recommendation Update

- **Objective:** Ensure that deleting a profile removes associated recommendations.
- **Preconditions:** Profile is uploaded and matched to jobs successfully.
- **Steps:**
    a. Upload a profile and verify job recommendations.
    b. Delete the uploaded profile.
- **Expected Result:** System removes the profile and its associated job recommendations.
- **Pass Criteria:** Recommendations are no longer available after profile deletion.

Test Case 12: Recommendation Filtering by Location

- **Objective:** Validate that candidates can filter job recommendations by location.
- **Preconditions:** Profile is uploaded, and job recommendations are available.
- **Steps:**
    a. Upload a profile.

b. Filter the job recommendations by specific location(s).

- **Expected Result:** Recommendations update to show only jobs in the selected location(s).
- **Pass Criteria**: Location-based filtering accurately reflects the candidate's preference.

Test Case 13: Recommendation Sorting by Relevance

- **Objective:** Ensure that job recommendations can be sorted by relevance.
- **Preconditions:** Profile is uploaded, and job recommendations are available.
- **Steps:**
    a. Upload a profile.
    b. Sort job recommendations by relevance.
- **Expected Result:** Recommendations are displayed in order of relevance to the candidate's profile.
- **Pass Criteria:** Sorting functionality works correctly, showing the most relevant jobs first.

Test Case 14: Profile Privacy and Data Security

- **Objective:** Verify that the system ensures privacy and security of uploaded profiles.
- **Preconditions:** Profile is uploaded successfully.
- **Steps:**
    a. Upload a profile.
    b. Attempt unauthorized access to the profile data.
- **Expected Result:** Profile data remains secure and inaccessible to unauthorized users.
- **Pass Criteria:** System enforces proper security measures to protect profile data.

Test Case 15: Profile Parsing and Matching Speed

- **Objective:** Measure the speed of profile parsing and job matching processes.
- **Preconditions:** Candidate has a profile ready for upload.

- **Steps:**
    a. Upload a profile.
    b. Record the time taken for parsing and matching.
- **Expected Result:** Profile parsing and job matching are completed within acceptable time limits.
- **Pass Criteria:** System meets performance benchmarks for parsing and matching speed.

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 that 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 a simple workflow. This may vary depending on the complexity of your application and the skill set/experience of the team involved in the integration.

1. **Development (16hr – 19hr)**
    a. Upload Profile - 1hr
    b. Parse using RChilli API – 1hr
    c. Read JSON and display recommendations on UI – 8hr
    d. Store information in Database – 2-3hr
    e. Unit Test cases – 2-3hr
    f. Bug fixing and optimization – 2-3hr
2. **QA – (5hr -7hr)**
    a. UAT – 1-1.5hr
    b. Executing Test cases – 4-5.5hr

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