**Introduction (Business Need):**

We are dedicated to recruiting top talent from the industry. We embrace diversity, equality, and inclusion in our workplace, ensuring that everyone feels valued and inspired.

Our **recruiter is** dependent on other supporting tools to search right resume for the given JD (job description) manually. It’s nightmare to search relevant resume for JD. They need to put extra effort and filter out candidates with many criteria.

On the other side **candidates**, whether internal or external, can explore job openings through the **Workday** portal using basic filters such as location, role etc.

At present, the portal lacks the intelligence to suggest/recommend the relevant candidates resumes to JD (job description). Same for the candidates, they are not able to search relevant job openings for their skills, leading candidates to potentially overlook opportunities or they struggle to discover them.

Objective is to design a solution which can be integrated with Workday portal in production for relevant resume and job recommendations.

**Challenge Description:**

In this challenge, you will build a system which can use existing resumes and Job Descriptions (JD) from **Workday** and perform these requirements:

* Recruiter/candidate user should **search resume/jobs using keywords** (prompts). They should have the capability to conduct **Google-like** keyword searches, allowing them to search using a combination of skills, experience, domain, and other relevant criteria.
* The ML based recommendation algorithms should effectively analyze search keywords resume/job listings to **generate relevant** and **meaningful suggestions**.
* **Note: Hackathon team must expose REST API to evaluate model efficiency. API contract would be provided in this document at the bottom.**

**Benefits:**

1. **Efficiency**: Streamlines the resume and job search process by presenting tailored opportunities, saving time for both recruiter and candidates.
2. **Personalization**: Provides personalized suggestions based on individual job description, role, skills, experience, and preferences improving the relevance of resume/job matches.
3. **Reduced Missed Opportunities**: Minimizes the likelihood of recruiter/candidates missing suitable opportunities, job openings, ensuring a comprehensive exploration of relevant candidate and job options based on their skills.
4. **User-Friendly Interface**: Improves the user experience with a user-friendly web interface, making the recruiter and candidate resume/job search and application process more intuitive and accessible.

**Use Cases:**

* **Mandatory => Resume Recommendation (for Recruitment Team):** Suggest the top relevant resumes based on job description keywords (prompts).
* **Optional (Good to have)=> Job Recommendation (for internal/external candidates):** Suggest the relevant jobs/roles based on skill set based on keywords (prompt) search.

**Assumptions**

* **Workday** portal functionalities are pre-existing in the system and that includes features such as the creation of accounts, the capability to upload resumes **PDF/Word** format**.**
* Users provide accurate and up-to-date information regarding their job description, skills, experience, and preferences.
* The system is designed to handle a growing number of users, job listings, and data without compromising performance.
* Focused effort is required on building recommendation engine which can be plugged-in with Workday.

**Prerequisite**

* Sufficient sample resumes and job Details (JDs) would be provided for the hackathon purpose to train data model.
* **Note: User should not use their own resumes and Job descriptions.**

**User Stories**

1. As a recruiter, I want to get resume recommendations based on keyword search.
2. As a candidate, I want to receive personalized job recommendations based on keyword search.
3. As a product owner, I aim for the software to be impartial, ensuring it does not exhibit bias based on gender, race or origin. (Just a guideline, only focus on recruiter/candidate users)

## Solution Package:

Following are the tasks that need to be executed as part of this challenge:

* Provide high level user stories implementation for the challenge.
* Provide a Design approach.
* Code that needs to be uploaded to the Git repo (Instructions for Git repo would be provided by core team).
* Provide playbook – Build, deployment and run steps.
* Automated test suite and coverage report.

**Details of each of the deliverables are as follows:**

#### User Stories

This document needs to capture the high-level user stories that are part of the platform design. It would be desirable to have roles/actors to be part of the user stories and scenarios that were considered.

#### Design Approach

The design document should detail out overall approach on solving the problem. This could include any class diagrams/sequence diagrams indicating the approach and a high-level traceability matrix.

#### Working code

The resultant application should have good quality and generic enough so that new rules can be added and/or removed easily and committed to respective Git repos that are assigned.

#### Playbook

Provide Playbook/Runbook for deploying the solution. Provide information towards how it should be deployed & monitored.

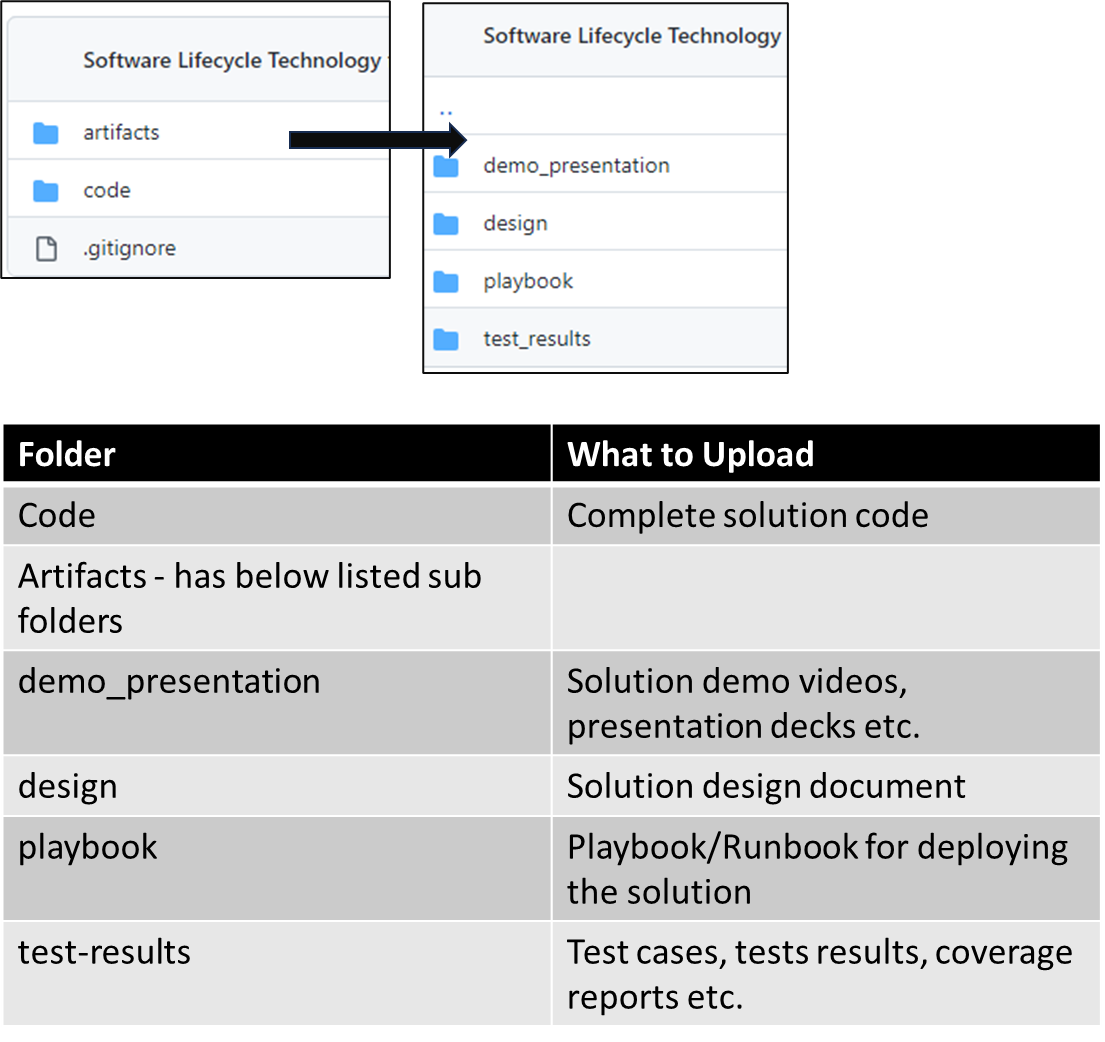
#### Automated Test Suite and Coverage Report

Test cases for various edge cases should be automated. Application needs to be tested as per the test cases and the results need to be captured along with the coverage report and any excel report and graphs.

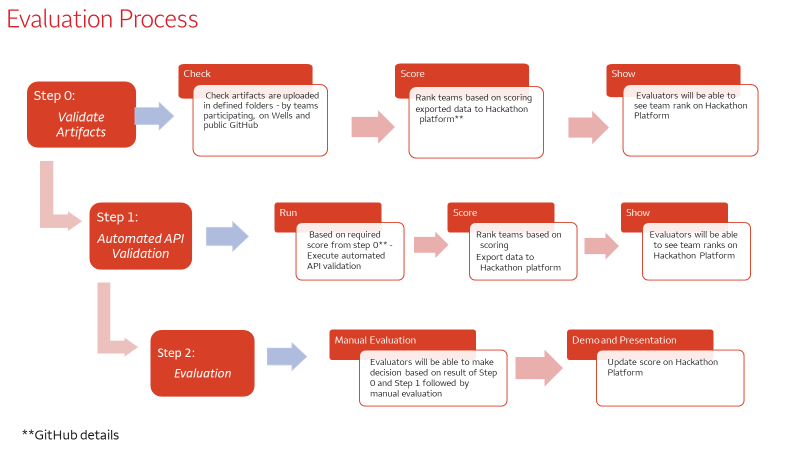
**Artifacts to be uploaded**

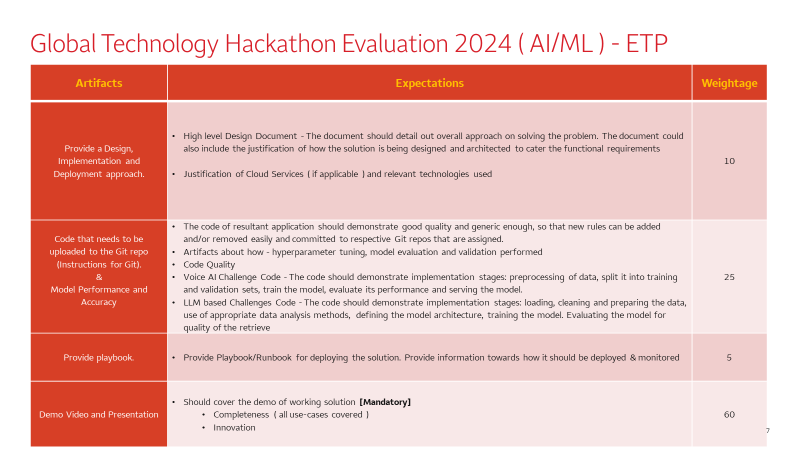
Below mentioned artifacts and code to be uploaded to Git repository following the instructions below. It is mandatory to adhere to these instructions to be eligible for evaluation.

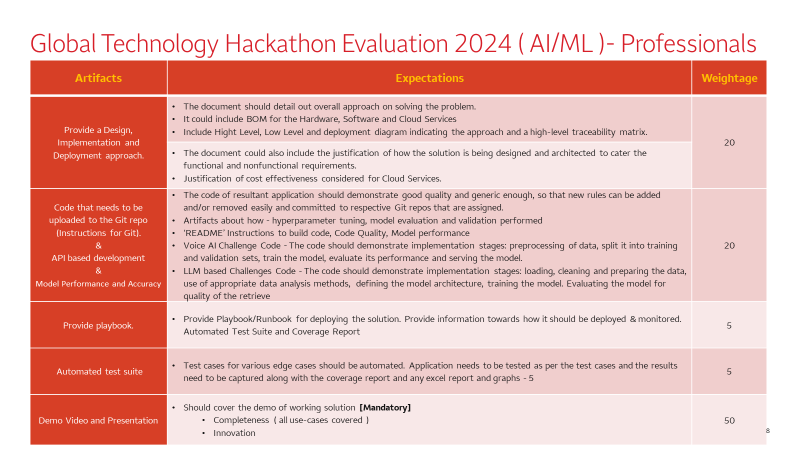
* Git repo has the following folder structure.
  + **Artifacts** 
    - **demo\_presentation**
    - **design**
    - **playbook**
    - **test\_results**
  + **Code**
* **Git folder structure**



#### Evaluation Approach







## FAQs

We recommend you refer to the common FAQs provided for the main challenge.

## REST API to Evaluate ML Model

Most important: Team provide Azure or GCP cloud storage folder/bucket/fileshare HTTPS path with write access. Evaluation team will add test data and run evalution tool.

“inputPath: “<<Place-holder>> “

**Note: This location will be used by evaluation team to add sample resumes/JD for testing purpose. Team has to create this storage ( GCP bucket/Azure Container or Fileshare on their Azure/GCP account and submit URL also with solution doc.**

| {  "context": "CheckColumn I Value",  "category": "resume",  "threshold": "Check Column G"  "noOfMatches": 3,  "inputPath": "<https://console.cloud.google.com/storage/browser/hackathontestdata2024>"  } | {  "count": 3,  "metadata": {  "confidenceScore": 0.9  },  "results": [  {  "id": 15891494,  "path": "15891494.pdf",  "score": 0.90  },  {  "id": 24833063,  "path": "24833063.pdf",  "score": 0.90  },  {  "id": 98513424,  "path": "98513424.pdf",  "score": 0.90  }  ],  "status": "success"  } | 200 | <https://resumejobsearch.free.beeceptor.com/search> | POST | [{"key": "Content-Type", "value": "application/json"}] |
| --- | --- | --- | --- | --- | --- |
| {  "context": "java software development",  "category": "job",  "threshold": 0.7,  "noOfMatches": 3,  "inputPath": "jobeDataSet"  } | {  "count": 8,  "metadata": {  "confidenceScore": 0.65  },  "results": [  {  "id": 12345,  "path": "jobDataSet/12345.pdf",  "score": 0.8  },  {  "id": 45678,  "path": "jobDataSet/45678.pdf",  "score": 0.75  },  {  "id": 45678,  "path": "jobDataSet/45678.pdf",  "score": 0.75  }  ],  "status": "success"  } |  | <https://resumejobsearch.free.beeceptor.com/search> | POST | [{"key": "Content-Type", "value": "application/json"}] |
| NA | {  "status": "healthy",  "dependencies": {  "modelAPIS": {  "model1": "online",  "model2": "offline"  },  "database": {  "connection": "available",  "responseTime": "12 ms"  },  "memory": {  "usage": "normal"  },  "cpu": {  "usage": ".5"  }  }  } |  | <https://resumejobsearch.free.beeceptor.com/ping> | GET |  |
| {  "context": "java software development",  "category": "inventory",  "threshold": 0.7,  "noOfMatches": 3,  "inputFilePath": "jobDataSet"  } | {  "status": "Bad Request Please check category should be either resume or job!"  } | 400 | <https://resumejobsearch.free.beeceptor.com/search> | POST | [{"key": "Content-Type", "value": "application/json"}] |

| **Functionality/API** | **category** | **Actor/User** |
| --- | --- | --- |
| **Mandatory**=> Sample Input:  {  "context": "Java J2EE developer 5 years experience",  "category": "resume",  "threshold": 0.7,  "noOfMatches": 3,  "inputPath": "[https://console.cloud.google.com/storage/browser/hackathontestdata2024](https://urldefense.com/v3/__https:/console.cloud.google.com/storage/browser/hackathontestdata2024__;!!F9svGWnIaVPGSwU!oncwmARouwCNTauH51mSt3lNu7XjHJ9wwqcIUiyEmkFJv2uQDQkZTYng2TMWUEvbPXG1wm4zk3wI1GsOs6ew0FB5YYE$) "  }  **Important Notes**:   * **All the parameters are mandatory.** * **The number of matches returned should match the limit “noOfMatches”. For e.g if your model returns 5 matches, return only top 3.**   **Sample Response**  {  "status":"success",  "count": 3  "metadata":{  "confidenceScore": 0.8  }  "results":  [  {  "id": 1  "score": 0.8  "path": "1.pdf"  },  {  "id": 2  "score": 0.75  "path": "2.pdf"  },  {  "id": 3  "score": 0.66  "path": "3.pdf"  }  **]**  **Good to have**  Job/Profile Recommendation Keyword search API:  {  "context": "Java J2EE developer 5 years experience",  "category": "job",  "threshold": 0.7,  "noOfMatches": 3,  "inputPath": "[https://console.cloud.google.com/storage/browser/hackathontestdata2024”](https://urldefense.com/v3/__https:/console.cloud.google.com/storage/browser/hackathontestdata2024__;!!F9svGWnIaVPGSwU!oncwmARouwCNTauH51mSt3lNu7XjHJ9wwqcIUiyEmkFJv2uQDQkZTYng2TMWUEvbPXG1wm4zk3wI1GsOs6ew0FB5YYE$)  }  **Sample Response**  {  "status":"success",  "count": 3  "metadata":{  "confidenceScore": 0.8  }  "results":  [  { | resume | Recruiter |
| **Good to have => Sample Input:**  Job/Profile Recommendation Keyword search API:  {  "context": "Java J2EE developer 5 years experience",  "category": "job",  "threshold": 0.7,  "noOfMatches": 3,  "inputPath": "[https://console.cloud.google.com/storage/browser/hackathontestdata2024”](https://urldefense.com/v3/__https:/console.cloud.google.com/storage/browser/hackathontestdata2024__;!!F9svGWnIaVPGSwU!oncwmARouwCNTauH51mSt3lNu7XjHJ9wwqcIUiyEmkFJv2uQDQkZTYng2TMWUEvbPXG1wm4zk3wI1GsOs6ew0FB5YYE$)  }  **Sample Response**  {  "status":"success",  "count": 3  "metadata":{  "confidenceScore": 0.8  }  "results":  [  {  "id": 12345  "score": 0.8  "path": "12345.pdf"  },  {  "id": 45678  "score": 0.75  "path": "45678.pdf"  },  {  "id": 34567  "score": 0.66  "path": "34567.pdf"  } | job | Candidate |

**Final Important Notes**:

* Hackathon team will provide set of sample resume data set, and actual test should be on golden set copy.
* Folder resume Data Set will have few thousands resume and 50+ jobs for testing. Participants must develop and test against sample resume and job description data set provided.
* **Location for Training Data:** <https://console.cloud.google.com/storage/browser/hackathon1415>
* **Location for Test Data**: (Team just has to test connectivity from their API to this folder/bucket location:<https://console.cloud.google.com/storage/browser/hackathontestdata2024>
* **GCP** 
  + **Location for Training Data:** [**https://console.cloud.google.com/storage/browser/hackathon1415**](https://console.cloud.google.com/storage/browser/hackathon1415)
  + **Location for Test Data** : (Team just has to test connectivity from their API to this folder/bucket location: [**https://console.cloud.google.com/storage/browser/hackathontestdata2024**](https://console.cloud.google.com/storage/browser/hackathontestdata2024)
  + **Python code to Download from GCP Bucket:**
  + **GCP Bucket key to access from programming languages Java/Python /Dot Net etc- You can download from this location:**
  + Python Code to download files from GCP bucket:

| **Python code to GCP:**  from flask import Flask, request, jsonify  import json  app = Flask(\_\_name\_\_)  # Dummy function for matching resumes to job descriptions using ML.  # In a real scenario, this would involve loading a trained model and comparing embeddings.  def match\_resumes\_to\_jobs(resume\_text, job\_description\_text):  # Logic for using ML model to compare texts.  # Placeholder for simplicity: returns True if there's a "match", False otherwise.  return True  @app.route('/submit\_resume', methods=['POST'])  def submit\_resume():  data = request.json  resume\_text = data['resume']  candidate\_name = data['name']    # Assuming job descriptions are stored and accessed somehow:  job\_descriptions = [{'job\_title': 'Software Developer', 'description': 'Job description for Software Developer'}]    matches = []  for job in job\_descriptions:  if match\_resumes\_to\_jobs(resume\_text, job['description']):  matches.append({'candidate\_name': candidate\_name, 'job\_title': job['job\_title']})    # Send resume & name to recruiter, job title & description to candidate here.  # For simplicity, just return the match information.  return jsonify(matches)  @app.route('/evaluate\_model', methods=['GET'])  def evaluate\_model():  # Logic to evaluate model efficiency.  # This could involve calculating metrics like precision, recall, etc., based on test data.  model\_efficiency = "Dummy model efficiency metrics."  return jsonify({"model\_efficiency": model\_efficiency})  if \_\_name\_\_ == '\_\_main\_\_':  app.run(debug=True) |
| --- |

**AZURE**

· **Location for Training Data:** [**https://hackathon1415.blob.core.windows.net/data?comp=list**](https://urldefense.com/v3/__https:/hackathon1415.blob.core.windows.net/data?comp=list__;!!F9svGWnIaVPGSwU!vsmDptmz61I4JwhupOp6xoXzjwckfqxeTIjGMoQyRX_puLDvSshngQe1QyKjzFe-G07iqn8ooVdYQDxPIDVyXu6wEsulLw$)

· **Location for Test Data** : (Team just has to test connectivity from their API to this folder/bucket location: [**https://hackathontestdata2024.blob.core.windows.net/data?comp=list**](https://urldefense.com/v3/__https:/hackathontestdata2024.blob.core.windows.net/data?comp=list__;!!F9svGWnIaVPGSwU!vsmDptmz61I4JwhupOp6xoXzjwckfqxeTIjGMoQyRX_puLDvSshngQe1QyKjzFe-G07iqn8ooVdYQDxPIDVyXu5rf4xm8Q$)

Azure Storage account name-**hackathon1415**

**Key:** Ay1jvnupCShhBmo5ky/j2IPeMPBCkIxYP8gZ6RVAdKi6RgBlMdWBoylkJ/67BDU5DzCQkwRComSV+AStxoGWGQ==

**Connection String:** DefaultEndpointsProtocol=https;AccountName=hackathon1415;AccountKey=Ay1jvnupCShhBmo5ky/j2IPeMPBCkIxYP8gZ6RVAdKi6RgBlMdWBoylkJ/67BDU5DzCQkwRComSV+AStxoGWGQ==;EndpointSuffix=[core.windows.net](http://core.windows.net/)

**Azure Access Storage from Java:**

<https://github.com/Azure/azure-sdk-for-java/blob/main/sdk/storage/azure-storage-blob/src/samples/java/com/azure/storage/blob/ListContainersExample.java#L10>

**Azure Access Storage from Python:**

<https://github.com/toddkitta/azure-content/blob/master/articles/storage/storage-python-how-to-use-blob-storage.md>

**Power Shell Script:**

Invoke-WebRequest -URI <https://hackathon1415.blob.core.windows.net/data?comp=list> -outFile response.xml

[xml]$cn = Get-Content .\response.xml

foreach ($bename in $cn.EnumerationResults.Blobs.Blob) {

Invoke-WebRequest -URI "$($bename.Url)" -outFile "$($bename.Name)"

Write-Host "$($bename.Url)"

}

**<<End of document>>**