API Configuration Documentation

Overview

To tackle the problem of building an SHL Assessment Recommendation System, we will break down the task into multiple phases. The goal is to develop a web application that provides recommendations for relevant SHL assessments based on job descriptions or queries, ensuring that it returns results with specific details like test name, URL, remote testing support, and more. Below is a detailed approach to solving the problem.

Step 1: Understanding the Problem Requirements

- **Input**: The system needs to handle natural language queries, job descriptions, or URLs that refer to job postings.
- **Output**: Based on the input, the system will recommend up to 10 relevant SHL assessments, each with attributes such as:
    - Assessment name and URL (linked to SHL's catalog)
    - Remote Testing Support (Yes/No)
    - Adaptive/IRT Support (Yes/No)
    - Duration
    - Test Type

Base Requirements

- **Assessment Name**
- **URL**: For linking to the SHL catalog
- **Remote Testing Support**: Whether the test can be conducted remotely
- **Adaptive/IRT Support**: Whether the test is adaptive or uses Item Response Theory (IRT)
- **Duration**: The time it takes to complete the test
- **Test Type**: The nature of the test (e.g., Cognitive, Personality, Skills Assessment)
- **Job Role Identification**: Extract the job title (e.g., "Java Developer", "Analyst") from the input.
- **Skill Identification**: Identify specific skills required for the role (e.g., "Python", "SQL", "JavaScript").
- **Time Constraints**: Extract any time-related requirements from the query (e.g., "within 40 minutes").

- **Assessment Type**: Extract any specific needs related to the test type, such as "Cognitive" or "Personality".

The system needs to serve as a web application where hiring managers can input queries or upload job descriptions. We can use tools like **Streamlit** or **Gradio** to quickly build a user interface (UI) for the application.

Required Endpoints

Web Application Features:

1. **Input Handling**: Users can input a natural language query or provide a job description (via text or URL).
2. **Processing the Query**: The backend processes the input query to extract meaningful information (skills, job role, etc.).
3. **Recommendation Display**: The system displays the top 10 relevant assessments in a tabular format with the required attributes (name, URL, remote testing, adaptive/IRT, duration, test type).
4. **API Endpoint**: Provide an API endpoint where users can query with JSON and receive results in a machine-readable format.

We can host the backend API on **Flask** or **FastAPI**, and the front-end UI can be built with **Streamlit**.

Step 6: Code Implementation

Below is a high-level overview of the code implementation:

json


```
{
  "assessment_name": "Java Development Skills Test",
  "url": "https://www.shl.com/solutions/products/java-developer-assessment",
  "remote_testing": "Yes",
  "adaptive_support": "Yes",
  "duration": "40 minutes",
  "test_type": "Skills Assessment"
}
```


Response Fields Explanation

For the /recommend endpoint response:

Field	Type	Description
url	String	Valid URL to the assessment resource
adaptive_support.	String	Either "Yes" or "No" indicating if the assessment supports adaptive testing
description	String	Detailed description of the assessment
duration	Integer	Duration of the assessment in minutes
remote_support.	String	Either "Yes" or "No" indicating if the assessment can be taken remotely
test_type	Array of Strings	Categories or types of the assessment

2. **Backend Code** (Flask or FastAPI example):

python

```
from fastapi import FastAPI
from pydantic import BaseModel
import spacy
from transformers import pipeline

app = FastAPI()

# Load language model for query processing
nlp = spacy.load('en_core_web_sm')
qa_pipeline = pipeline("question-answering")

class Query(BaseModel):
    query: str

@app.post("/get_assessments/")
```

```
def get_recommendations(query: Query):
    # Process the query using NLP and LLMs
    doc = nlp(query.query)
    # Extract job role, skills, time, etc.
    # Then, filter and rank assessments based on relevance
    relevant_assessments = recommend_assessments(doc)
    return {"recommendations": relevant_assessments}

def recommend_assessments(doc):
    # Logic for filtering and ranking assessments based on the query
    return [{
        "assessment_name": "Java Developer Skills Test",
        "url": "https://www.shl.com/solutions/products/java-developer-assessment",
        "remote_testing": "Yes",
        "adaptive_support": "Yes",
        "duration": "40 minutes",
        "test_type": "Skills Assessment"
    }]
```

3. **Frontend Code** (Streamlit Example):

```
import streamlit as st
import requests

st.title("SHL Assessment Recommendation System")

user_input = st.text_input("Enter your job description or query:")

if user_input:
    response = requests.post("http://<backend_url>/get_assessments/", json={"query": user_input})
    recommendations = response.json().get("recommendations", [])

    if recommendations:
```

```
        st.write("Recommended Assessments:")
        for rec in recommendations:
            st.markdown(f"[{rec['assessment_name']}]({rec['url']})")
            st.write(f"Remote Testing: {rec['remote_testing']}")
            st.write(f"Adaptive/IRT Support: {rec['adaptive_support']}")
            st.write(f"Duration: {rec['duration']}")
            st.write(f"Test Type: {rec['test_type']}")
    else:
        st.write("No relevant assessments found.")
```

Links:

Website: https://pasupuletisivakira.wixsite.com/shl-assessment-rec-1

Github: https://github.com/2000080186/PASUPULETI-SIVA-KIRAN

https://github.com/2000080186/PASUPULETI-SIVA-KIRAN/blob/main/Fast%20API%20shl.py

Api : http://127.0.0.1:8000/

Path: C:\\Users\\PASUPULETI SIVAKIRAN\\.vscode\\project

api file:

```
from fastapi import FastAPI
from pydantic import BaseModel
import spacy
from transformers import pipeline
import json

# Initialize FastAPI app
app = FastAPI()

# Load spaCy NLP model for text processing
nlp = spacy.load('en_core_web_sm')
```

```
# Example: Using Hugging Face's Transformers to load a pre-trained question-answering model (you can also use GPT or similar models)
qa_pipeline = pipeline("question-answering")

# Example of a SHL catalog data (this would usually be fetched from a database or an external file)
shl_catalog = [
  {
    "assessment_name": "Java Development Skills Test",
    "url": "https://www.shl.com/solutions/products/java-developer-assessment",
    "remote_testing": "Yes",
    "adaptive_support": "Yes",
    "duration": "40 minutes",
    "test_type": "Skills Assessment"
  },
  {
    "assessment_name": "Python Developer Test",
    "url": "https://www.shl.com/solutions/products/python-developer-assessment",
    "remote_testing": "Yes",
    "adaptive_support": "No",
    "duration": "60 minutes",
    "test_type": "Skills Assessment"
  },
  {
    "assessment_name": "Cognitive Ability Test",
    "url": "https://www.shl.com/solutions/products/cognitive-ability-assessment",
    "remote_testing": "Yes",
    "adaptive_support": "Yes",
    "duration": "30 minutes",
    "test_type": "Cognitive Assessment"
  }
```

```
]

# Define request body model
class Query(BaseModel):
    query: str

# Endpoint for getting relevant assessments based on a query
@app.post("/get_assessments/")
async def get_assessments(query: Query):
    # Process the query using spaCy to extract entities (this can be expanded as needed)
    doc = nlp(query.query)

    # Here we can use NLP or a model to extract meaningful info (e.g., skills, role, duration)
    # For the sake of simplicity, we'll assume it extracts key words like "Java", "Python", etc.

    # Example of keyword extraction logic (you can enhance this)
    keywords = [token.text for token in doc if token.pos_ in ["NOUN", "PROPN"]]

    # For simplicity, we're matching keywords with test names or types
    relevant_assessments = []
    for assessment in shl_catalog:
        if any(keyword.lower() in assessment["assessment_name"].lower() for keyword in keywords):
            relevant_assessments.append(assessment)

    # Return results in JSON format
    return {"recommendations": relevant_assessments}
```