SPRINT 3

TEAM ID	PNT2022TMID22144
PROJECT NAME	Skill Based Job Recommender

CHAT BOX

CODE:

```python
import streamlit as st
import streamlit.components.v1 as components
import pandas as pd
import numpy as np
import base64,random
import re,os
from ftfy import fix_text
from nltk.corpus import stopwords
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors
import time,datetime
from pyresparser import ResumeParser
from pdfminer3.layout import LAParams, LTTextBox
from pdfminer3.pdfpage import PDFPage
from pdfminer3.pdfinterp import PDFResourceManager
from pdfminer3.pdfinterp import PDFPageInterpreter
from pdfminer3.converter import TextConverter
import io,random
from streamlit_tags import st_tags
import streamlit.components.v1 as stc
from PIL import Image
import sqlite3
from Courses import ds_course,web_course,android_course,ios_course,uiux_course,resume_videos,interview_videos
import plotly.express as px
from pathlib import Path
import hashlib

script_location = Path(__file__).absolute().parent
os.chdir(script_location)
```

```python

def make_hashes(password):
    return hashlib.sha256(str.encode(password)).hexdigest()

def check_hashes(password,hashed_text):
    if make_hashes(password) == hashed_text:
        return hashed_text
    return False

conn = sqlite3.connect('data.db')
c = conn.cursor()
def create_usertable():
    c.execute('CREATE TABLE IF NOT EXISTS userstable(username TEXT,password TEXT)')


def add_userdata(username,password):
    c.execute('INSERT INTO userstable(username,password) VALUES (?,?)',(username,password))
    conn.commit()

def login_user(username,password):
    c.execute('SELECT * FROM userstable WHERE username =? AND password = ?',(username,password))
    data = c.fetchall()
    return data


def pdf_reader(file):
    resource_manager = PDFResourceManager()
    fake_file_handle = io.StringIO()
    converter = TextConverter(resource_manager, fake_file_handle, laparams=LAParams())

    page_interpreter = PDFPageInterpreter(resource_manager, converter)
```

```
    page_interpreter = PDFPageInterpreter(resource_manager, converter)
    with open(file, 'rb') as fh:
        for page in PDFPage.get_pages(fh,
                                      caching=True,
                                      check_extractable=True):
            page_interpreter.process_page(page)
            print(page)
        text = fake_file_handle.getvalue()

    # close open handles
    converter.close()
    fake_file_handle.close()
    return text

def show_pdf(file_path):
    with open(file_path, "rb") as f:
        base64_pdf = base64.b64encode(f.read()).decode('utf-8')
    # pdf_display = f'<embed src="data:application/pdf;base64,{base64_pdf}" width="700" height="1000" type="application/pdf">'
    pdf_display = F'<iframe src="data:application/pdf;base64,{base64_pdf}" width="700" height="1000" type="application/pdf"></iframe>'
    st.markdown(pdf_display, unsafe_allow_html=True)

def course_recommender(course_list):
    st.subheader("**Courses & Certificates🎓 Recommendations**")
    c = 0
    rec_course = []
    no_of_reco = st.slider('Choose Number of Course Recommendations:', 1, 10, 4)
    random.shuffle(course_list)
    for c_name, c_link in course_list:
        c += 1
        st.markdown(f"({c}) [{c_name}]({c_link})")
```

```
        st.markdown(f"({c}) [{c_name}]({c_link})")
        rec_course.append(c_name)
        if c == no_of_reco:
            break
    return rec_course


def searchbox():
    prefinal = []
    skill_ip = []
    skill_ip = str(st.text_area('Enter the Skills and press cntrl + enter'))
    butt = st.button("search")
    if butt:
        prefinal = skill_ip.split('\n')
        recommender(prefinal)

def rsl(rsd):
    resskill = rsd['skills']
    recommender(resskill)

def convertTuple(tup):
    # initialize an empty string
    str = ''
    for item in tup:
        str = str + item
    return str

def recommender(rskl):
    jskills = []
    jskills.append(' '.join(word for word in rskl))
    org_name_clean = jskills
```

```
    org_name_clean = jskills

    stopw  = set(stopwords.words('english'))
    df1 =pd.read_csv('job_final.csv')
    df1['test']=df1['Job_Description'].apply(lambda x: ' '.join([word for word in str(x).split() if len(word)>2 and word not in (stopw)]))

    def ngrams(string, n=3):
        string = fix_text(string) # fix text
        string = string.encode("ascii", errors="ignore").decode() #remove non ascii chars
        string = string.lower()
        chars_to_remove = [")","(",".","|","[","]","{","}","'"]
        rx = '[' + re.escape(''.join(chars_to_remove)) + ']'
        string = re.sub(rx, '', string)
        string = string.replace('&', 'and')
        string = string.replace(',', ' ')
        string = string.replace('-', ' ')
        string = string.title() # normalise case - capital at start of each word
        string = re.sub(' +',' ',string).strip() # get rid of multiple spaces and replace with a single
        string = ' '+ string +' ' # pad names for ngrams...
        string = re.sub(r'[,-./]|\sBD',r'', string)
        ngrams = zip(*[string[i:] for i in range(n)])
        return [''.join(ngram) for ngram in ngrams]

    vectorizer = TfidfVectorizer(min_df=1, analyzer=ngrams, lowercase=False)
    tfidf = vectorizer.fit_transform(org_name_clean)


    def getNearestN(query):
        queryTFIDF_ = vectorizer.transform(query)
        distances, indices = nbrs.kneighbors(queryTFIDF_)
        return distances, indices
```

```
        return distances, indices

    nbrs = NearestNeighbors(n_neighbors=1, n_jobs=-1).fit(tfidf)
    unique_org = (df1['test'].values)
    distances, indices = getNearestN(unique_org)
    unique_org = list(unique_org)
    matches = []
    for i,j in enumerate(indices):
        dist=round(distances[i][0],2)

        temp = [dist]
        matches.append(temp)
    matches = pd.DataFrame(matches, columns=['Match confidence'])
    df1['match']=matches['Match confidence']
    df11=df1.sort_values('match')
    df2=df11[['Position', 'Company','Location','url']].head(10).reset_index()
    # st.table(df2)

    list_template = """
    <div style='color:#fff'>
    <h2>{}</h2>
    <h3>{}</h3>
    <h4>{}</h4>
    </div>
    """
    for i in range(len(df2)):
        jt = df2.loc[i, "Position"]
        cp = df2.loc[i, "Company"]
        lc = df2.loc[i, "Location"]
        jurl = df2.loc[i,"url"]
        st.markdown(list_template.format(jt,cp,lc),unsafe_allow_html=True)
```

```
        st.markdown(list_template.format(jt,cp,lc),unsafe_allow_html=True)
        with st.expander("Apply for job"):
            stc.html(jurl)


st.set_page_config(
   page_title="Job Recommender",
)
def run():
    st.title("Job Recommender")
# st.markdown('''<h4 style='text-align: left; color: #d73b5c;'>* Upload your resume, and get smart recommendation based on it."</h4>''',
#             unsafe_allow_html=True)

    st.sidebar.markdown("# Job Recommender")
    activities = ["Sign Up","Log In","Search For Job Recommendation","Upload Resume"]
    choice = st.sidebar.selectbox("Choose: ", activities)
    if choice == "Sign Up":
        st.subheader("Create an Account")
        new_user = st.text_input('Username')
        new_passwd = st.text_input('Password',type='password')
        components.html('''
                <style>
                #place{
                    posistion:fixed;
                    bottom:0;
                }
                </style>
                <div id="place">
                    <script>
                        window.watsonAssistantChatOptions = {integrationID: "0bb96b92-4e98-44c7-9dab-3a5fe2ff8562",region: "au-syd", servic
                        setTimeout(function(){const t=document.createElement('script'); t.src="https://web-chat.global.assistant.watson.app
```

```
                        setTimeout(function(){const t=document.createElement('script'); t.src="https://web-chat.global.assistant.watson.app
                    </script
                </div>
                ''',height=600,)
```