WolfTrack4.0
Documentation

WolfTrack 4.0 helps to plan and organize job applications in a chronological order so that you can easily track job applications, get expert suggestions, and successfully get your desired company. We store job applications, job profiles, locations, salaries, dates, notes, and more!

Project Structure:

Our project mainly constitutes of the following folders:

kotlin
Copy code
```
WolfTrack/
├── Controller/
├── DAO/
├── Templates
├── Static
├── Unit Testing
```
Controller:

The controllers handle the backend-to-front-end connection and are responsible for routing the web pages to their respective methods. Home.py mainly handles the routing of web pages in this project. Controller also contains the main functionalities that help build this project. For example, send_email.py, ResumeParser.py, cron_job_slave.py, etc.

Templates, Static:

The Templates folder contains the frontend HTML pages of the project, whereas the Static folder mainly consists of the CSS files, JavaScript files, and images used in the project.

Unit Testing:

This folder contains the unit test cases that are used to test each functionality in the system.

Data Access Objects (DAO):

The DAO layer connects the backend of the application to the database (AWS RDS).

Functionalities:

home.py: This file is mainly used to route the web pages and call the required functionalities. Each function in this file helps to fetch details from the frontend and use that data by passing it to the respective called functionality.

jd_scrapper.py: This file scrapes jobs from indeed.com and saves it in the database. This data is then used to suggest the user a list of jobs based on their profile.

send_email.py: This functionality sends an email notification to the user that he/she has added a job/company to its respective job list (Wishlist, In-process, Applied, Offers). The user fills a job application form which consists of various fields. These details are sent to the user by an email notifying that the job is successfully added to the list. This email and addition to the list can be used by the user for future reference.

ResumeParser.py: We are using docx2txt, pypdf2, and cv2 libraries to parse Word documents, PDFs, and images, respectively. These will parse and convert the documents into text format for further processing and analysis.

activity_controller.py : The "activity_controller" file contains a Flask-RESTful resource class named "Activity" that handles HTTP methods for an activity-related functionality. Users are required to be logged in to access these endpoints, ensuring authentication. The class supports GET, POST, PUT, and DELETE requests, echoing received data with a "you sent" message and a status code of 201.

Chat_gbt_pipeline.py: The "chat_gpt_pipeline" script streamlines the process of obtaining detailed resume critiques using OpenAI's GPT-3.5 Turbo. It first converts PDF resumes to plain text for analysis. By instructing the AI model with specific criteria, it generates comprehensive feedback, making it a convenient tool for resume optimization and enhancing job-seeking prospects.

Word Cloud Creation: In order to scan the job description, we would need to ensure we consider all the important keywords related to a person's resume. Creation of a word cloud is going to help us flash those keywords for a quicker scan.

Resume Score Calculator: We are using the count vectors to store the output we captured from the word cloud analysis, and then we use Cosine Similarity function to achieve a similarity score between the two texts of the Resume and the Job Description.

Cron Job: The automated deadline email functionality allows the user to get email reminders one day before the event deadline. It is implemented using Python crontab. This is used to implement cron_job_slave.py, which iterates over the events and sends the email if the deadline is for the next day. There is another cron job cronjob_reminder.py which runs to send an email about the upcoming job deadlines which are stored in our database.

Send_profile: The send_profile functionality allows the user to send their profile to the specified email address of either a colleague, an expert, or any other related person. The email lists the companies which are being waitlisted by the user and details of which companies the user has applied to, the offers he/she has received, and the in-progress status of the companies. This helps the user to get suggestions from other people regarding their job applications. This also sends the user's resume along with his details as an attachment.

Upload and View Resume functionality: The resume uploading and viewing functionality allow the user to choose to upload a specific resume version from their computer. The uploaded file is stored into a directory called "resume" in the "Controller" folder. The function accepts all types of files to be uploaded, be it .docx or .pdf. The application can access the file from the "files" dictionary based on the request given. A user can choose to view his uploaded resume so that he can know through which resume version he applied to that job. The user can also use that resume for the resume parser and analysis functionality as mentioned above. This functionality also supports multiple file extensions now.

Main_login.py: This file contains a login landing for admin as well as user login.