# 1. Objective

This project aims to imitate some simple operations of a job searching website. It provides users with up-to-date job post information and supports job searching and filtering. It also allows companies to post and manage new job opportunities decently. The resources are collected from Indeed. The developer hopes to realize similar functions as a normal job searching website.

(Add an overview of background knowledge, include Indeed introduction, screenshot of Indeed list page)

# 2. Overview

This project is built up with 4 important components, which can be listed as follows:

1) ***web crawler***: used to collect job posts from [Indeed](https://sg.indeed.com/?r=us) routinely

2) ***front-end pages***: provide user interfaces for job searching and result displaying

3) ***back-end API***: realize the logic for web pages, retrieve and proceed with user inputs while interact with databases to collect and return the results

4) ***databases***: used to store all the useful information (roles, users, jobs, etc.) and provide quick searches for job posts (Add document DB)

# 3. Infrastructure

The overall project is written in Python, and supported by Flask.

The web crawler uses Selenium to simulate user login and to retrieve the static / dynamic contents from the original website. Once the contents are collected, they are proceeded locally with Beautiful Soup (bs4).

The front-end pages are displayed as html files with CSS models. Bootstrap is chosen for page rendering. The results of interaction between front-end and back-end APIs are reflected in web pages with the help of Jinja2 template rendering engine, which is provided by Flask.

The back-end logics are realized with Flask, a lightweight Python framework. The visiting activities are recorded by sessions, which are supported by flask\_session.

Two types of database systems are applied in this project. MySQL is selected for storing all the information, including roles, accounts and job posts (forward indexed). Sqlalchemy is applied to interact with MySQL through Python script. Meanwhile, since the main function of a job searching site is the search for jobs, Elasticsearch is adopted to support inverted indexing of the job posts in order to accelerate the searching process.

# 4. Detailed Design

## 4.1. Web Scrawler

Selenium is used to automate the process of logging into the website and retrieving the latest information. Initially, the scrawler returns all the job posts with keywords “software”, “IT” and “developer”. The following information are extracted from each job card (a job post record from Indeed):

|  |  |
| --- | --- |
| Column Name | Description |
| URL | Link to the detailed description of a job post |
| Job\_title | Name/Title of the job |
| Company\_name | Name of the company who releases this job post |
| Salary | Claimed salary on the job post |
| Release\_date | Date when the job post was released |
| Description | Job-snippet of an indeed job post |

Table 1. key information for web scrawler

Specifically, a selenium.webdriver object is instantiated with web driver Chrome. After that, a search URL is generated with desired keywords (title, company, link, date, desc) with the help of urllib. The resulting webpages are scrapped one by one by the webdriver object, and returned to a BeatifulSoup object for finer extraction.

BeautifulSoup is imported to process with html files locally. It will extract all the key information listed in Table 1. All the extracted information will be collected in a Python dictionary which will later be transferred into a record in jobposts table in the database.

The project is initialized with all the related job posts within 30 days (init\_jobposts()), and updated once a day (update\_jobposts()).

## 4.2. Front-end UI

The front-end UI would mainly be supported by Bootstrap, including the CSS configurations and Javascripts. The UIs can be divided into the following parts: home.html, register.html, login.html, job-manage.html, edit-jobpost.html, user-manage.html. The rought layouts of the above files are shown in Graph 1.

Password

User name

Confirm password

Email

User type

Favor

button

(job title with hyperlink)

(company name)

(salary)

(release date)

Alerts

Confirm button

Search results

(or recommended posts)

User

Info

Filters

Search bar

Subgraph 1.1. home.html Subgraph 1.2. register.html

Company info

Create a new job post

Filters for current posts

Current job posts

delete

(job title with hyperlink)

(salary)

(release date)

Confirm button

Password

Email

Alerts

Subgraph 1.3. login.html Subgraph 1.4. job-manage.html

Delete

(user id, user name, user type, user permissions listed as columns)

Filter

User List

Confirm button

Description

Salary

Job title

Subgraph 1.5. edit-jobpost.html Subgraph 1.6. user-manage.html

Graph 1. Rough layouts of front-end pages

As a detailed description, home.html is a public webpage for all kinds of users (tourists, job seekers, companies, admins). A search bar is used to collect the searching keyword, and a set of filters are used to select/reorganize some of the records from the search results. The search results are organized in pages (ten records for a page). Besides, registered users can favor some of the job posts for recording and comparing. The favored posts and past searching records help build up the user portraits, and would be used be recommend users with related job posts. The recommendations will be listed if no search is conducted by a registered user.

Register.html and login.html serve as a universal platform for all kinds of users to register and login.

Job-manage.html and edit-jobpost.html are restricted to registered companies. Companies can choose to add a new job post, view the posted jobs and delete some of the posts. A filter is provided to select some of the posted jobs from the entire list. While the scrapped data from Indeed has their detailed descriptions linked as separate webpages, the manually added job posts would have their detailed descriptions stored as json objects (which can be supported by both MySQL and ElasticSearch) and be displayed in a rather simple way (titles + paragraphs).

User-manage.html is restricted to administrators. All the registered users would be listed in the user list. Administrators can see the name and type of the users and choose to delete some of them (which is dangerous!). An administrator cannot delete other administrators.

## 4.3. Back-end API

### 4.3.1. job searching, filtering and sorting

This API is mainly designed for job searching. First of all, check if any user has logged in. If yes, retrieve the user information from table “users” in MySQL with the “user\_id” in current session. Fill the information into “user info” in home.html (subgraph 1.1.). After that, refresh the webpage and show 10 recommended job posts. These recommended job posts are selected from the keywords collected from the user’s latest searches. To be more specified, the split words (分词结果？) from ElasticSearch analyzer of each search are stored as a json object, with the keyword and its frequency as key and value. The 3 most frequently-chosen words are selected to generate the recommendations. In order to do this, the json object is translated into a python dictionary, with its key ordered as a max-heap of their frequencies. The heapify process, along with picking up the 3 max elements, takes O(N) times. Although it is not so efficient to generate the recommendations, the update of the keywords can be done in O(1) times, which happens more frequently.

The searching process begins upon pressing the confirm button beside the search bar. The searching keyword would be analyzed by ElasticSearch and help locate the targets. At the same time, the keywords’ analyzing results would be updated to the user’s info.

After each search by a user/tourist, the results would be shown in a table. Only the first 300 records would be displayed in order to control the space cost. These records would be stored as a list of python dictionaries (excluding the description but involving the score provided by ElasticSearch). The filter works on certain keys of the dictionary, and filtering would take O(N) time. The sorting process is implemented with dual-pivot quicksort, which takes O(NlogN) time on average. The sorting functions are provided in app/utils.py.

### 4.3.2. register

This API uses the request class from flask to retrieve the payload of a post request from register.html. The payload should include a form with the following data: name, role, email, password, repassword.

{

“name” : “John”,

“role” : “jobseeker”,

“email” : “[123123@qq.com](mailto:123123@qq.com)”,

“password” : “11111111”,

“repassword” : “11111111”

}

Sample of a post request payload (as a form)

The API would interact with MySQL to ensure the username and email are unique for each account. At the same time, the password would be compared with “repassword” to make sure there’s no difference. All the errors are collected as a list and result in an alert in register.html. If no error happens, a new account would be created and automatically logged in.

|  |  |
| --- | --- |
| Common Error | Alert Message |
| Conflicted username | This user name has been used. Pls enter another valid name. |
| Conflicted email | This email address has been used. Pls enter another valid address. |
| Wrong “repassword” | The confirmed password does not match the password. Pls check again. |

Common errors and their alert messages

### 4.3.3. log in

This API works for login.html. Similar to 4.3.2., this API gets a post request with email and password, find out the user with the given email and check the correctness of password. Occurance of empty account or wrong password would be recorded as a list and displayed as alerts in login.html.

### 4.3.4. job post editing

This API is called when a company click on the “create a new job post” button, and the user must pass the permission check. All the information provided in edit-jobpost.html would be sent as a post request, while the company name woud be automatically acquired from the current session. The creation date would be set as the current date.

### 4.3.5. job managing and user managing

Job managing API and user managing API follow a similar logic. First, the page permission should be checked. After that, the respective user information would be displayed in the managing page, and the job posts/users list would be displayed in a table. Different from that of a job search, the source of the job post list / users list is MySQL, considering that many / all of the records in a table should be returned. The records are stored in a list of python dictionaries (would it be better to use a view of the table?). The filtering and sorting processes are the same as that of the search results for job searching (4.3.1.).

## 4.4. Database Systems

Two database systems are used for this project. Elasticsearch is adopted specifically for job searching, and includes one main table/type: jobposts. MySQL is used for general purposes, and include four main tables: roles, users, jobposts, favored\_jobposts.

The metadata of table/type jobposts in ElasticSearch is described in Table 2. Each of the file in ES represents a job post.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| Post\_id (is it needed?) | Integer | (primary key) Unique id for the file |
| Job\_title | Text | Name/Title of the job |
| URL | Keyword | Link to the detailed description of a job post; nullable |
| Company\_name | Text | Name of the company who releases this job post |
| Salary\_min | Integer | Loweset claimed salary on the job post |
| Salary\_max | Integer | Highest claimed salary on the job post |
| Release\_date | Date | Date when the job post was released |
| Description | Text | Job-snippet of an indeed job post or self-added descriptions by the company. |

Table 2. metadata of type “jobposts”

Specifically, “job\_title” and “description” are used for matching and scoring. URL is an optional field that links to the original, detailed description on Indeed. If URL is set null, click on the URL would lead to a simple webpage showing all the information in Table 2 (except for URL) as keys and values (designed for company-generated posts).

The Role table in MySQL stores the basic information for different roles in the project. Only registered roles would be recorded in this table, which include jobseekers, companies and admins. The metadata of table Role is shown in Table 3.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| Role\_id | Integer | (primary key) Unique id for a role |
| Name | Varchar(80) | Name of the role |
| Permissions | Integer | A union set of permissions for the role |

Table 3. metadata of table “roles”

All the permissions are listed as follows:

|  |  |  |
| --- | --- | --- |
| Notation | Representation | Description |
| a | 0x01 | job searching, filtering, sorting |
| b | 0x02 | job post favoring |
| c | 0x04 | creating a new job post |
| d | 0x08 | managing the job posts of his own company |
| e | 0x10 | managing the users |

Table 4. permissions and their representations

Table Users in MySQL stores the information of each registered users, their metadata described in Table 5.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| User\_id | Integer | (primary key) Unique id for a user |
| Name | Varchar(100) | Name of the user |
| Email | Varchar(100) | Email of the user |
| Role\_id | Integer | (foreign key) role id of the user |
| Password | Varchar(20) | Password for the user’s account |
| Search\_history | Json | Splitted words stored with frequencies |

Table 5. metadata of table “users”

Different kinds of users are assigned with different permissions. Table 6 shows the relationship between roles and permissions.

|  |  |  |
| --- | --- | --- |
| Role Name | Permissions (sum) | Permitted operations(in notation) |
| Tourist(unregistered) | 0x01 | a |
| Jobseeker | 0x03 | a, b |
| Company | 0x0f | a, b, c, d |
| Admin | 0x1f | a, b, c, d, e |

Table 6. relationship between roles and permissions

Table jobpost in MySQL is very much similar to table/type jobpost in ES. The metadata of table “jobpost” is described in table 7.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| Post\_id | Integer | (primary key) Unique id for the file |
| Job\_title | Varchar(100) | Name/Title of the job |
| URL | Varchar(200) | Link to the detailed description of a job post; nullable |
| Company\_name | Varchar(100) | Name of the company who releases this job post |
| Salary\_min | Integer | Loweset claimed salary on the job post |
| Salary\_max | Integer | Highest claimed salary on the job post |
| Release\_date | Timestamp | Date when the job post was released |
| Description | Text | Job-snippet of an indeed job post or self-added descriptions by the company. |

Table 7. metadata for table “jobpost”

“favoredjobpost” is stored for each user and provides easy access for job posts tracing and comparing. The metadata of this table is shown in table 8.

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| Record\_id | Integer | (primary key) Unique id for the favored post |
| User\_id | Integer | (foreign key) User\_id for the user who favored this post |
| Post\_id | Integer | (foreign key) Post\_id for the job post favored |

Table 8. metadata for table “favoredjobpost”

# 5. Testing Plan

# 6. Work Estimate

20 days in total.

1) 2 days: design the web scrawler

2) 3 days: register and login UI design, as well as the respective back-end support. Need to get familiar with the basic components and templates of Bootstrap.

3) 4 days: design for the job searching page and back-end logics, including the search, display and reorganizing of job posts.

4) 2 days: support user favoring function for job searching part.

5) 4 days: design for the jobpost manage and user manage page and back-end logics.

6) 4 days: user recommendations design