Job Stats System Guide

Web Scraper: The Jobstats Web Scraper Is defined inscrapebydate.py. It visits Indeed.com at the following URL `"https://www.indeed.com/jobs?q=computer+science&fromage=" + str(iFrom) + "&toage=" + str(iTo) + "&sort=date&filter=0&limit=50&start=` where it iterates through each page collecting 50 job listings per page. The iFrom and iTo parameters tell the scraper how old each listing collected should be (in days). A multiple of 50 is appended to the end of the url to to iterate through the pages. The scraper visits each non sponsored listing on the page and collects the indeed id from the url, the date the listing was posted, the job title, the location, the company, and the job description. The Scraper compares indeed ids to ones in the database to ensure only new listings are saved.

The webscraper utilizes a logging system which records how many listings were scanned, how many new listings were saved, and what errors were encountered.

Map: The map, it's sidebar, and associated features are defined in MapContainer.js and MapAndSideBarContainer.js. MapContainer defines the map, markers, and heatmap while MapAndSideBar defines the side bar and performs the api calls to gather data for the map. The map depends on the Google Maps React package (Note this is not the same as Google Map React)

MapAndSideBarContainer:
GoogleMapsContainer (exported from MapContainer.js) requires six props.

style: any inline css that you want to use on the map.

city_names: an array of strings to be used as city names for the map markers (these are retrieved from col.json in location_data via an api request in getCitiesAndCityNames() inside MapAndSideBarContainer)

cities: an array of city objects as defined in col.json. This is obtained the same way that city_names is.

center: a google maps latlng object that sets the visual focus of the map. This is set in the file MapAndSideBar by handleSideBarClick when an entry in the side bar is clicked.

map_style : another css struct that styles only the google map itself, just pass this the same height and width used in style.

top_cities: an array of google weighted location objects obtained from an api call to the top_locations end point in createTopCities inside the MapAndSideBarContainer file.

MapContainer Quirks:
In the render method of Map Container an if statement checks to make sure that both state.top_cities and state.cities have been filled by the api calls in MapAndSideBarContainer. If these arrays are empty or incomplete the component will fail to generate it's markers and/or the heatmap.

The file heatmap.js Inside the Node module Google Maps React had to be modified in order to get weighted heatmap locations working. Be sure that you have override the default file with the one in static/JS. Otherwise the heatmap will not work and may prevent the map from rendering.

HomeBar:
Home.js defines much of the site's routing. Here, the routing to each default tile, location tile, custom tile, as well as the sign in page and map is defined. The file also renders the navbar at the top of the page and displays a message to portrait phone users to turn their phones to landscape for a better experience.

Migrations: If you want to add new default/location tiles to the site, you will need to create a migration file for them. QUIRKS: make sure you list a dependency on the previous migration (see bottom of existing migrations for examples). If you need to rerun migrations you will have to change their number to one that has not been run before and change the corresponding dependencies.

Insights: Insights are generated on a per tile basis, with the dataset upon which the insights are found being dependent on the parameters of the tile. When a tile is instantiated, DataHandler determines the top 10 skills for the set of filters, titles, companies (and possibly locations) provided, and creates a JSON dataset with the top skills and a corresponding array describing the frequency of each skill for each day (or week) since data collection began. Insights are then found by calling methods of insights.py with the created dataset. Each method uses pandas to perform analysis on the dataset, and returns a hash with information such as the type of insight, the skill(s) for which the insight was found, a template string that presents the insight in a digestible way to the user, and a confidence score signifying statistical significance. Here is a summary of the methods of insights.py:

- get_trending_up: Finds which skill has had most positive rate of growth since the start of data collection
- get_trending_down: Finds which skill has had the most negative rate of growth since the start of data collection
- get_skill_location: Finds the skill for which the greatest difference exists between the city where it's seen the most demand, and the city with the next closest demand (this method is not called for location tiles)
- get_dominant_skill: Finds the skill with the greatest average percent appearance in job listings since the beginning of scraping
- get_correlation: Returns the pair of skills for which the greatest pairwise correlation exists

Data Handler: Uses Elasticsearch to query our entire database of job listings and retrieve statistical data. Contains methods for returning trend data, bar data and determining top skills, locations. These methods take in filters, companies, titles, and locations; they will only consider data with those specifications. When creating a DataHandler object it must be given a start value as a time object. This DataHandler object will then only consider data after that time.

Daily Update: On the production server a cron job runs a script every day which scrapes job listings and then updates the top skills and insights of every tile in the database by calling the appropriate method on each tile.

Tiles: A tile is an object which contains filters, companies, titles, and locations which are then used to generate data for that tile. It also contains top skills and insights which are displayed on the tile page. Tiles come in two varieties: custom and standard. Custom tiles have an extra user_id field used to identify the owner of the tile. Standard tiles have an extra style field which is currently unused but is intended to house style info which could be used to give each standard tile a unique look. A homepage for any given user will display all standard tiles except location tiles, as well as any custom tiles created by that user.

REST API Docs

Job Data API

api/trend_data

GET
Returns job histogram data in the form the form of x and y values to be plotted on a line graph.
params:

1. keywords: keywords to be plotted
2. filters: only data with all of these keywords in the description will be considered, format: array of strings
3. companies: only data from any of these companies will be considered, format: array of strings
4. titles: only data with any of these job titles will be considered, format: array of strings
5. locations: only data from any of these locations will be considered, format: array of strings
6. start: only data after this time will be considered, format: ISO8601 formatted timestamp
7. raw: should be '1' if raw data required, otherwise y values will be in the form of percents
8. period: data will be aggregated over periods of this length, valid values: 'month', 'week', 'day'

response:
{"x": [[...],[...]], "y": [[...],[...]], "raw": "0", "filters": [], "period": "week", "companies": [], "titles": [], "locations": []}

api/bar_data

GET

Returns job data in the form the form of y values to be plotted on a bar graph.
params:

1. keywords: keywords to be plotted
2. filters: only data with all of these keywords in the description will be considered, format: array of strings
3. companies: only data from any of these companies will be considered, format: array of strings
4. titles: only data with any of these job titles will be considered, format: array of strings
5. locations: only data from any of these locations will be considered, format: array of strings
6. start: only data after this time will be considered, format: ISO8601 formatted timestamp
7. raw: should be '1' if raw data required, otherwise y values will be in the form of percents

response:
{"y": [[...],[...]], "raw": "0", "filters": [], "companies": [], "titles": [], "locations": []}

api/top_skills

GET

Returns the skills with the most occurrences and their occurrence counts
params:

1. filters: only data with all of these keywords in the description will be considered, format: array of strings
2. companies: only data from any of these companies will be considered, format: array of strings
3. titles: only data with any of these job titles will be considered, format: array of strings
4. locations: only data from any of these locations will be considered, format: array of strings
5. start: only data after this time will be considered, format: ISO8601 formatted timestamp
6. count: number of skills to be returned, defaults to 10

response:
{"top_skills": [{"key": "python", "doc_count": 369}, {"key": "c", "doc_count": 358}, {"key": "java", "doc_count": 299}, {"key": "ios", "doc_count": 297}, {"key": "c++", "doc_count": 290}, {"key": "automation", "doc_count": 237}, {"key": "debugging", "doc_count": 179}, {"key": "qa", "doc_count": 162}, {"key": "sql", "doc_count": 158}, {"key": "macos", "doc_count": 156}]}

api/top_locations

GET

Returns the locations with the most occurrences and their occurrence counts
params:

1. filters: only data with all of these keywords in the description will be considered, format: array of strings
2. companies: only data from any of these companies will be considered, format: array of strings
3. titles: only data with any of these job titles will be considered, format: array of strings
4. start: only data after this time will be considered, format: ISO8601 formatted timestamp
5. count: number of locations to be returned, defaults to 10

response:
{"top_locations": [{"city": "Austin, TX", "doc_count": 96}, {"city": "Seattle, WA", "doc_count": 33}, {"city": "San Francisco, CA", "doc_count": 19}, {"city": "San Diego, CA", "doc_count": 9}, {"city": "Sacramento, CA", "doc_count": 7}, {"city": "Pittsburgh, PA", "doc_count": 6}, {"city": "Portland, OR", "doc_count": 4}, {"city": "New York, NY", "doc_count": 4}, {"city": "Denver, CO", "doc_count": 3}, {"city": "Milwaukee, WI", "doc_count": 3}, {"city": "Cupertino, CA", "doc_count": 2}, {"city": "Dallas, TX", "doc_count": 2}, {"city": "Chicago, IL", "doc_count": 1}, {"city": "Orlando, FL", "doc_count": 1}, {"city": "Richmond, VA", "doc_count": 1}]}

api/job_listings

GET
Returns the most recent job listings
params:

6. filters: only data with all of these keywords in the description will be considered, format: array of strings
7. companies: only data from any of these companies will be considered, format: array of strings
8. titles: only data with any of these job titles will be considered, format: array of strings
9. locations: only data from any of these locations will be considered, format: array of strings
10. start: only data after this time will be considered, format: ISO8601 formatted timestamp
11. raw: should be '1' if raw data required, otherwise y values will be in the form of percents
12. count: number of listings to be returned, defaults to 10

response:
{"job_listings": [{"indeed_id": "4772b50403c4817e", "posted_date": "2019-06-08T00:00:00", "title": "Siri - Software Engineer", "location": "Santa Clara Valley, CA", "company": "Apple", "description": "...” }, ...], "filters": [], "companies": ["apple"], "titles": [], "locations": []}

api/get_json_file

GET
Returns the contents of a json file on the server

params:

1. name: the name of the file as a string, do not include file extension
2. category: the folder containing the file as a string, valid values: 'insights', 'top_skills', 'location_data'

response:
the exact contents of the named json file or an error if not found

User API

user_info

GET
Returns the first and last name of the currently signed in user
params:
a valid session cookie must be sent along with this request in the Cookie request header
response:
{'first': ‘first_name’, 'last':’bob’, 'signed_in': True}

logout

POST
Logs out the currently signed in user
params:
a valid session cookie must be sent along with this request in the Cookie request header
response:
{'signed_in': False}

Tile API

tiles

GET
Returns a tile
params:

1. name: the name of the tile as a string

response:
{"tile": {"id": 2, "filters": [], "locations": [], "companies": ["apple"], "titles": [], "blacklists": [], "whitelists": [], "title": "Top Skills for Apple", "insights": [{"city": "austin", "type": "Location Insight",

"ratio": 12.0, "score": 5, "skill": "qa", "insight": "qa has been dominant in austin, with 1100% more jobs than the next closest city. This represents 5% of the total market for this skill."}, {"type": "Correlation", "score": 3, "insight": "Demand for c and c++ is tightly linked, with a correlation of 0.685253", "skill_one": "c", "skill_two": "c++", "correlation": 0.6852532344384941}, ...], "top_skills": [{"key": "python", "doc_count": 369}, {...},...], "name": "apple"}}

custom_tiles

GET
Returns all custom tiles for the currently signed in user
params:
    a valid session cookie must be sent along with this request in the Cookie request header
response:
{"custom_tiles": [...]}

POST
Creates a custom tile
params:
    a valid session cookie must be sent along with this request in the Cookie request header

1. filters: only data with all of these keywords in the description will be considered for this tile, format: array of strings
2. companies: only data from any of these companies will be considered for this tile, format: array of strings
3. titles: only data with any of these job titles will be considered for this tile, format: array of strings
4. locations: only data from any of these locations will be considered for this tile, format: array of strings
5. title:the title of the custom tile as a string

response:
{'tile': {...}, 'success': True}

PUT
Updates a custom tile
params:
    a valid session cookie must be sent along with this request in the Cookie request header, the session must match the owner of the custom tile whose name is provided

1. name: the name of the custom tile to be updated
2. filters: only data with all of these keywords in the description will be considered for this tile, format: array of strings
3. companies: only data from any of these companies will be considered for this tile, format: array of strings

4. titles: only data with any of these job titles will be considered for this tile, format: array of strings
5. locations: only data from any of these locations will be considered for this tile, format: array of strings
6. title:the title of the custom tile as a string

response:
{'tile': {...}, 'success': True}

DELETE
Deletes a custom tile
params:
a valid session cookie must be sent along with this request in the Cookie request header, the session must match the owner of the custom tile whose name is provided
1. name: the name of the custom tile to be updated

response:
{'success': True}