Tripalium

Motor de búsqueda para trabajos

Julio 2021

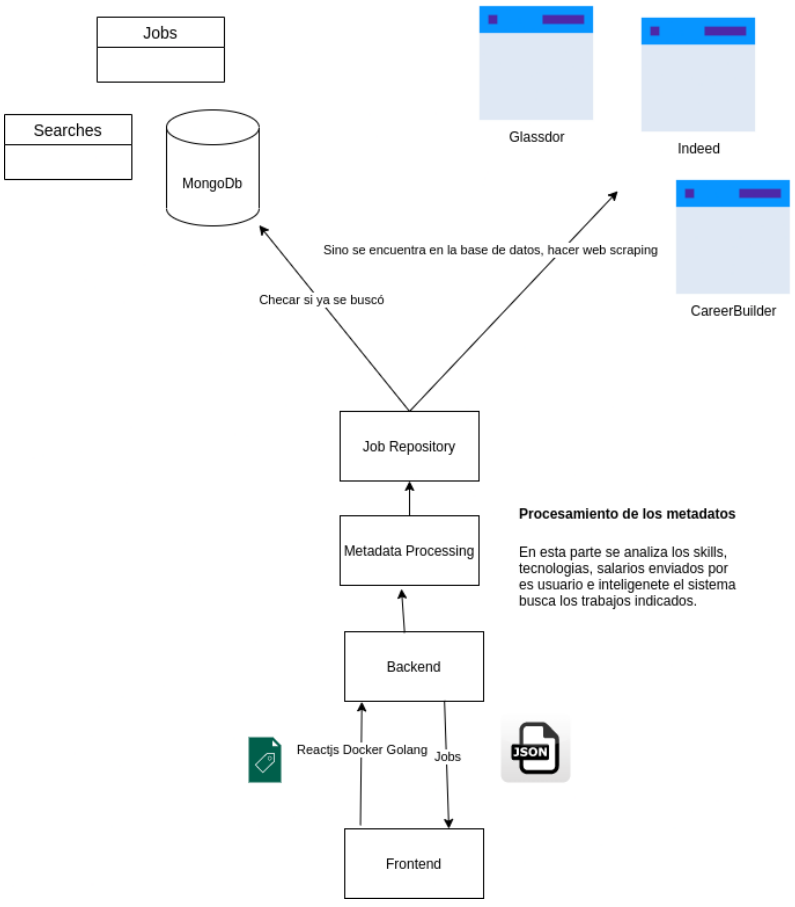
Análisis de grandes volúmenes de información

Mauricio Antonio Martínez Martínez
17121039

Descripción del proyecto

Siempre que te encuentras en la situación de encontrar un trabajo y te sientas abrumado por todos los diferentes sitios web que brindan información dentro de los diferentes requisitos y habilidades, los años de experiencia y los salarios pueden causar molestias para encontrar el adecuado para ti. Este motor de búsqueda está destinado a recopilar toda la información de los proveedores de trabajos más importantes y mostrarte los más adecuados para ti dependiendo de tus inputs.

Diagrama



Estructura Base Datos

En realidad usando NoSQL con mongodb no hay una estructura en sí pero estas son las colecciones que se encuentran y unos de sus datos más comunes.

Jobs	
title	string
enterprise	string
link	string
location	string
salary	float

Metadata	
keywords	string[]

Adquisición de Datos

Para obtener todos los empleos más importantes del mercado opté por hacer web scraping en los sitios de trabajos más famosos que puede encontrar: Indeed, Glassdoor y CareerBuilder. Debido a que linkedin require de autenticación para poder acceder a la página y de que usa “Client Site Rendering” los datos nos están en tiempo de compilación sobre templates sino que desde el cliente se hace la petición y posteriormente los despliega en la plataforma, lo cual hace el web scraping algo poco probable.

En los sitios mencionados agarré los datos más relevantes de cada trabajo y su descripción completa osea que si en un search encontré 10 trabajos otras 10 peticiones se hacen para recabar las descripciones completas.

Alamacenamiento

Como base de datos opté por usar una NoSQL ya que la diferencia en cuanto a la información entre página y página puede variar, la representación de la información se puede hacer bastante flexible en un esquema no relacional. En cuanto a las colecciones existen 2 la de trabajos y metadatos, si el usuario decide buscar una tecnología,habilidad o salario parecido al de otra búsqueda, esta queda guardada a parte para que futuras peticiones no tengan que hacer web scraping sino directamente consultar la base de datos.

Algoritmo

A gran escala no será óptimo el siguiente programa pero por fines de demostración está implementacion me parece la adecuada.

```python
class ScraperManager():
    """logic to fetch in all plataforms the same query"""

    def __init__(self, keywords):
        query = "-".join(keywords)
        country = "mx"
        url_glassdor = f'https://www.glassdoor.com.{country}/Empleo/{query}-empleos-SRCH_KO0,6_KE7,16.htm'
        query = "+".join(keywords)
        url_indeed = f"https://www.indeed.com/q-{query}-jobs.html?vjk=6b819f3191c070f1"
        location = "Mexico"
        url_career = f"https://www.careerbuilder.com/jobs?utf8=%E2%9C%93&keywords={query}&location={location}"

        self.career_builder = CareerBuilderScrepe(url_career)
        self.glassdoor = GlassdorScrape(url_glassdor)
        self.indeed = IndeedScrape(url_indeed)

    def main_scraping(self):
        with concurrent.futures.ThreadPoolExecutor() as executor:
            res_1 = executor.submit(self.indeed.scrape_jobs)
            res_2 = executor.submit(self.glassdoor.scrape_jobs)
            res_3 = executor.submit(self.career_builder.scrape_jobs)

            return res_1.result() + res_2.result() + res_3.result()
```

Scrapper Manager es una clase con la finalidad de ejecutar código en diferentes hilos, con ello logro hacer peticiones a la para y no esperar a que una termine, ideal para múltiples páginas que no dependen una de otra.

```python
def scrape_jobs(self):
    for job in self.data:
        title = job.find(True, {"class": "jobtitle"})
        url = self.DOMAIN + title["href"] if title else None

        salary = Util.get_att(
            job.find(True, {"class": "salaryText"}), "text")
        salary_qty = Util.get_salaries(salary)
        location = Util.get_att(
            job.find(True, {"class": "location"}), "text")
        urgent = bool(job.find(True, {"class": "urgentlyHiring"}))
        summary = job.find(True, {"class": "summary"})
        aspects = [Util.get_att(asp, "text") for asp in summary.findAll(
            "li")] if summary else None
        # description_html = self.scrape(url, {"id": "jobDescriptionText"})
        job_description = {"title": str(title.text).replace('\n', '') if title else None,
                           "salary": salary,
                           "link": url,
                           "salary_qty": salary_qty,
                           "location": location,
                           "aspects_summary": aspects,
                           "urgent": urgent,
                           # "preview": f"{self.DOMAIN}{href}",
                           # "description": description_html
                           }
        self.jobs.append(job_description)
```

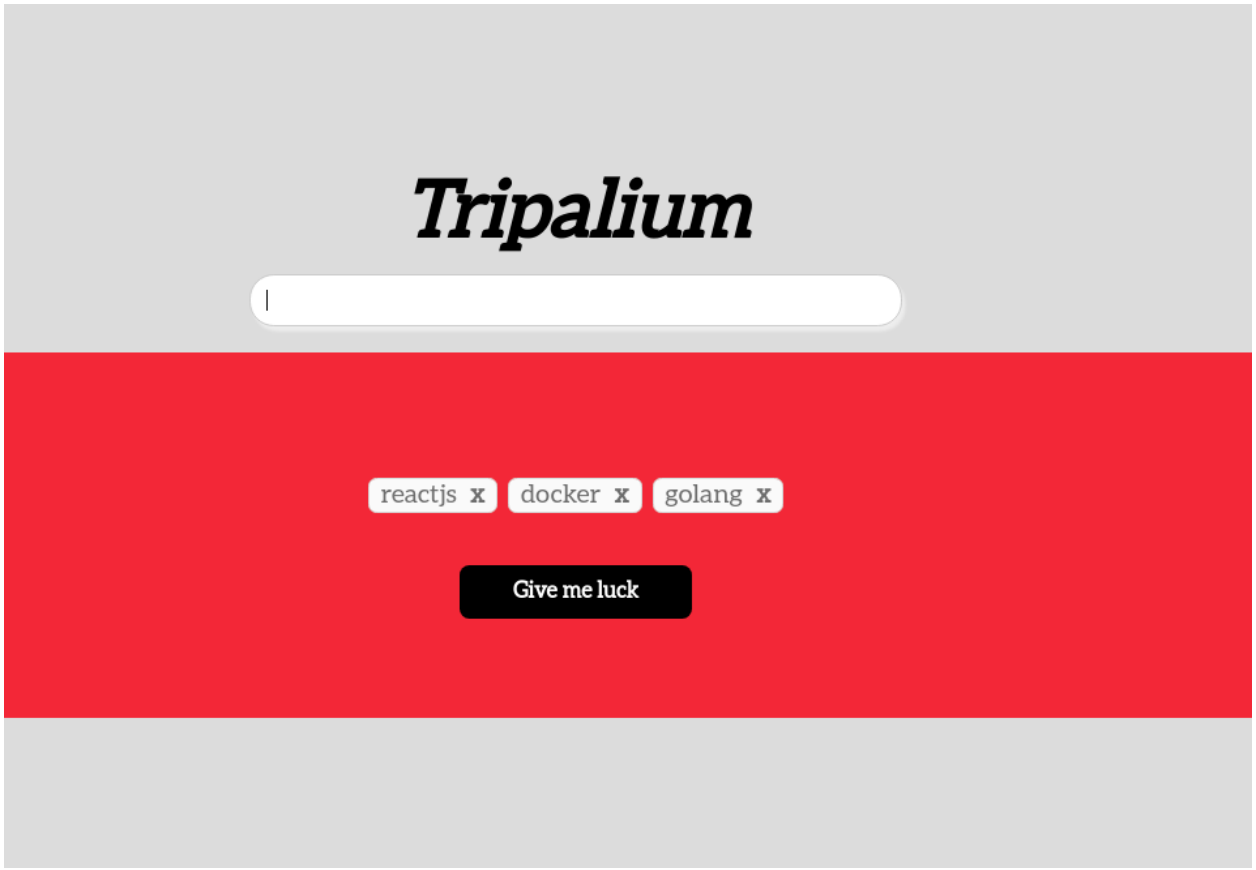
Cada página hace su propia implementación del cómo obtiene la información ya que la estructura por obvias razones cambia, en este ejemplo Indeed busca por tags ya previamente analizados en la página para ponerlos en un arreglo y posteriormente ser almacenados y enviados al usuario.

Todo está en un paradigma orientado a objetos, por lo que una clase abstracta que force a cada implementación de cada página a usar ciertos métodos comunes entre ellos.

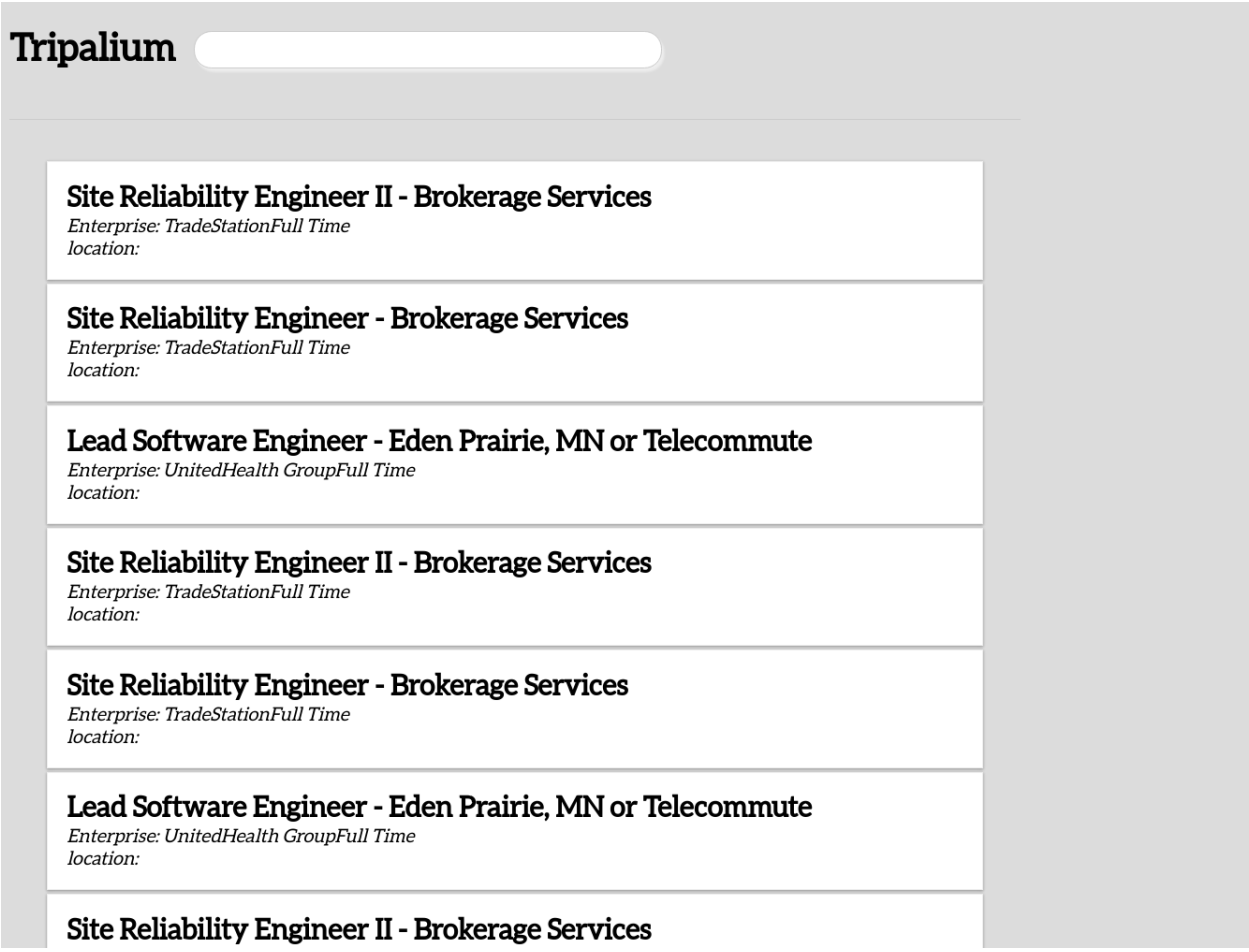
```python
class Scraper(metaclass=abc.ABCMeta):
    DOMAIN = None
    data = []
    args_scraper = {}
    jobs = []
    with_headers = False

    def __init__(self, url):
        if not url:
            raise Exception("Missing scraping url.")
        print(f"Scrapping url: {url}")
        data = self.scrape(
            url, self.args_scraper)
        self.data = data

    def scrape_jobs_descriptions(self):
        with concurrent.futures.ThreadPoolExecutor() as executor:
            """with executor.map you pass the array of futures and returns the results
                still running concurrently"""
            contents = executor.map(self.scrape_description, self.jobs)
            for index, content in enumerate(contents):
                self.jobs[index]["content"] = content

    @abc.abstractmethod
    def scrape_description(self):
        pass

    @abc.abstractmethod
    def scrape_jobs(self):
        pass

    def scrape(self, url, args, tag=True):
        try:
            headers = {
                "headers": {
```

Visualización

Primeramente el usuario entra a la página y como cualquier motor de búsqueda convencional solo un text field dónde deberá de insertar lo que quiere buscar, en este caso es metadatos de sobre tecnologías, habilidades etc que el usuario desee.



Una vez iniciada la búsqueda, se checará primeramente si no existen estos tags en el base de datos si existen busca por filtro y análisis probabilísticos la opción que mejor satisfaga dicha búsqueda el algoritmo prevé anteriores búsquedas de un mismo usuario y en base a su historial hace filtros de descarté para que una la información sea más específica y dos no tenga que ver los mismos resultados, esto hace que la respuesta del servidor sea diferente con cada usuario y su historial. A continuación un ejemplo de la respuesta.



Complejidad e impacto

En la parte técnica de la obtención de los datos sobre las distintas páginas es la parte más complicada ya que la estructura de estas puede cambiar y la forma de obtenerla usando hilos con concurrencia hace el proceso bastante ágil y eficiente pero inclusive la lógica detrás de escenas puede llegar a ser bastante complicado si se quiere ser más específico con la obtención de los datos.

Por otro lado el análisis lleva un poco más de facilidad solo es jugar con el json que se regresa y ajustarlo a las necesidades del usuario tomando en cuenta sus anteriores

búsquedas, la cantidad de veces que se repite una metadato(este lleva un peso mayor) etc., si tiene su magia pero sigue siendo algo más manejable que la obtención de los datos.

Conclusiones

En la actualidad la mayoría de los recién egresados y titulados, buscan trabajo ya sea en su localidad o en línea, pese a las actuales circunstancias el trabajo remoto llegó para quedarse siendo una competencia ya no por localidad sino mundial para ver quien se lleva el puesto, tener una herramienta que facilite saber que empleo es el indicado para tí veo que es de bastante peso para estos usuarios.