JobScraper

The Automatic Job Posting Recommender

- Chethan U Mahindrakar (002646783)
- Sayeed Ahmed (002191535)

Problem Statement and Idea

Problem Statement:

Job seekers need an efficient way to track job postings across Indeed. Manually checking each time is time-consuming and can lead to missed opportunities.

Solution:

Develop a web scraping application that scrapes job postings from Indeed, processes the data, and stores it in a database. Create an API that allows users to interact with the database and filter job postings based on their resume. Develop a Scala.js web interface that enables job seekers to search for job postings that are relevant to them.

Use Cases

Use Case 1: Job seeker looking for new opportunities

Actor: Job seeker

Action: The web scraping application regularly extracts job postings from various job portals, including Indeed, and stores the data in a database. The job seeker uses the Scala.js web interface to search for job postings and filter them based on their resume..

Reaction: Job seeker would be able to quickly and easily find job openings that match their skills and interests without having to manually search

~~**Use case 2: Company looking to monitor their competitors' hiring strategies**~~

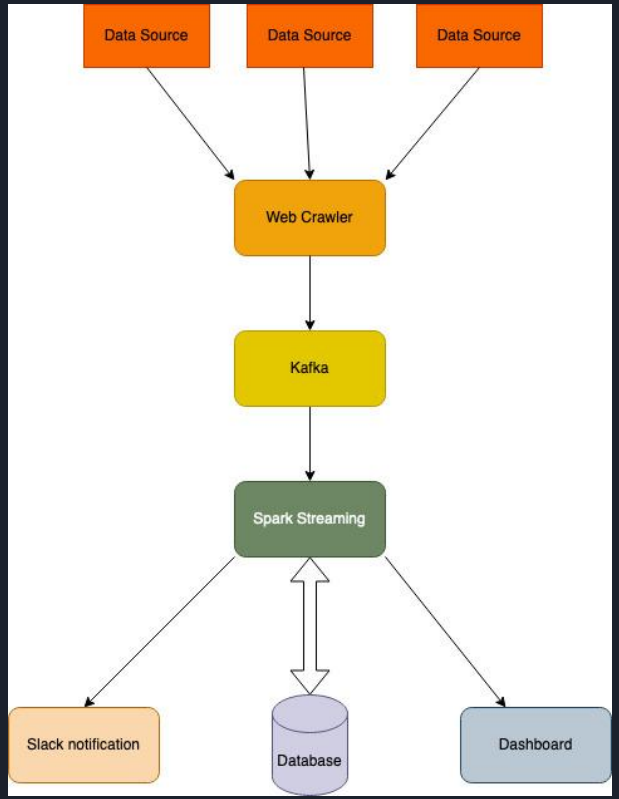
~~Actor: Company~~

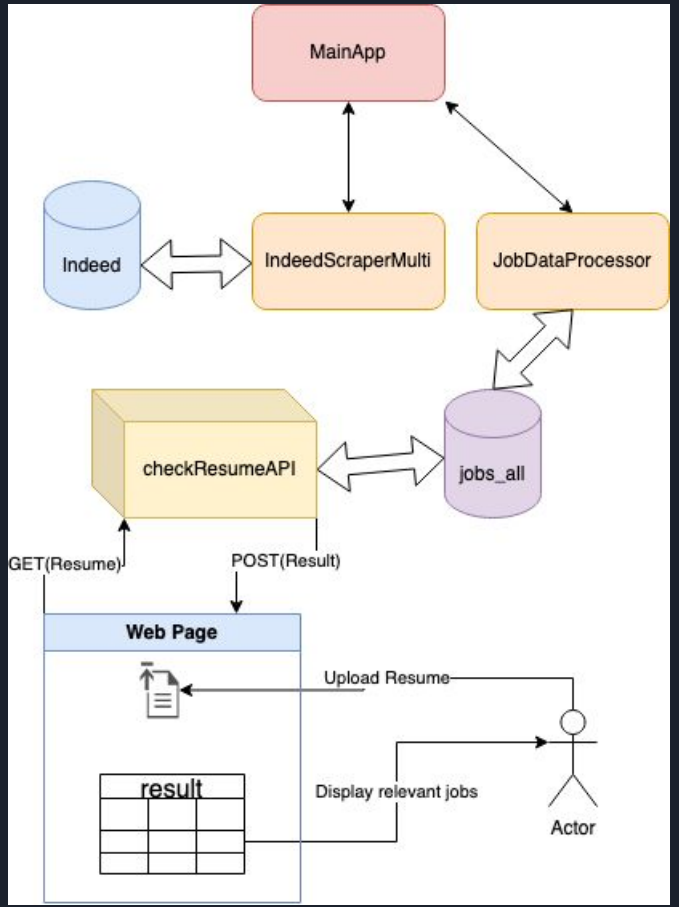
~~Action: The company would use the application to extract job postings from their competitors' websites and analyze the data to identify trends and patterns in their hiring strategies.~~

~~Reaction: The company would gain insights into their competitors' hiring strategies, allowing them to adjust their own recruitment strategies as needed.~~

Methodology

Proposed





Final

Methodology Changes

- Data Source Enhancement: Initially, we considered scraping multiple websites for data collection. However, due to the varied structures of each website, creating a universal scraper proved challenging. As a result, we shifted our focus to exclusively scrape Indeed, streamlining the data extraction process.
- Web Scraping Enhancement: As we shifted our focus to exclusively scrape Indeed, we faced the challenge of collecting ample data from a single source. To address this, we adopted a multithreading approach, enabling us to manage multiple scraper instances simultaneously. Each instance handled distinct parameters, allowing us to efficiently gather more data and optimize the extraction process.
- Resume based recommendation - Pick the top job postings with the highest match rate to the latest job postings fetched.

Data Source

- Our web scraper can efficiently generate approximately 1,800 rows of raw scraped data every 3 hours. Although we could increase this rate, doing so would raise the likelihood of being flagged or blocked. With a focus on maintaining a balance between data collection and risk mitigation, we have opted for this rate.
- Our Spark processor effectively processes the raw data, retaining about 92% of it through filtering, resulting in ~1,656 usable data rows.
- Despite the challenges encountered, we have successfully achieved close to the targeted amount of data for processing, demonstrating the effectiveness of our data collection and processing strategy.

THE API

- Hosted Locally. Endpoint - http://localhost:8001/check-resume
- Developed using SpringBoot Framework - In Scala
- Apache PDFBox library: "PDDocument" and "PDFTextStripper".
- Apache OpenNLP library to tokenize the input text, tag parts of speech in the text, and extract relevant keywords - typically nouns and adjectives. (Used over Job Description)
- API Response generated with (job_title, job_location, job_posting_link, percentage_match)
- Used the MVC pattern in the architecture.

THE INTERFACE

- Built a webpage using Scala.js and deployed it locally using http-server.
- Created an input field for users to upload their resumes and a select field for choosing job categories.
- Generated a POST request to the server with the uploaded resume and selected category using FormData API.
- Used XMLHttpRequest to send the POST request to the server and extract the response text. Parsed the response as a JavaScript object or array using JSON.parse method.
- Generated an HTML table element using document.createElement and populated it with job listings returned by the server. Styled the HTML table using CSS to improve visual appeal.
- Provided a valuable service to job seekers by allowing them to easily find job listings that match their skills and interests.

Milestones/Sprints

Week 1: Develop web scraper and implement data extraction from company websites.

Week 2: Process and analyze the data using Spark, database integration

Week 3: Implement the API for job recommendation

Week 4: Implement the dashboard using scala.js

~~Week 3: Implement Kafka for real-time processing and message queuing.~~

~~Week 4: Develop the Slack bot for user notifications and integrate it with the application.~~

Week 5: Optimize the system, conduct user testing, and finalize the project for submission.

Scala Programming and Code Repository

- Web scraping application: Scala
- Database: MySQL
- ~~Kafka integration: Scala using Kafka APIs~~
- ~~Slack bot: Scala for now / Any language that supports the Slack bot API, such as Python or JavaScript~~
- Spark: Scala
- Dashboard: ScalaJS
- API: Spring Boot, Scala
- Code repository: https://github.com/sayeedahmed01/JobScrapper

- Languages used:
  - Scala - 100%

Acceptance Criteria

Acceptance criteria -- mark if they were satisfied and show your actual result. For example, for A/C of 2 seconds response time, you actually achieved 1.3 seconds. Show with a check mark and "1.3"

- Extraction of Job Postings: The web scraper application should be able to extract job postings from a company's website with a success rate of at least 80%.
    - Achieved : Yes
    - We have a success rate of about 96% (4 in every 200 job posting are corrupted)
- Data Processing and Analysis: Spark should be able to process and analyze job postings data with a processing speed of at least 500 rows per second.
    - Achieved: Yes
    - The spark based dataprocessor is able to process ~800+ records in a second
- Periodic Retrieval and Storage: The application should be able to run periodically to retrieve new job postings and update the database with a success rate of at least 90%.
    - Achieved: Yes
    - The scraping application generates ~1800 rows of data every 4 hours out of which we have a success rate of ~92%

Goals

- To provide a convenient tool for job seekers to monitor multiple company websites for job postings in real-time.
  - The application provides a recommendation based job posting browser, with the data being updated at a satisfactory rate of every 4 hours.
- Understanding of Web scraping, Apache Kafka, Apache Spark, Data Streaming, Slack Developer API's.
  - Through the project we gained a in depth understanding of the Scala language, Web Scraping, Selenium, Data Streaming and additionally Scala JS and Spring Boot.
- Provide insights through analysis on the Job Market.
  - Unable to satisfy.